2007

Pattern topology for content management system

Mahdī Bāzargān
San Jose State University

PATTERN TOPOLOGY FOR CONTENT MANAGEMENT SYSTEM

A Thesis

Presented to

The Faculty of the General Engineering Program

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Mehdi Bazargan

December 2007

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346

APPROVED FOR THE GENERAL ENGINEERING PROGRAM

Dr. Mohamed E. Fayad

Dr. Dan Harkey

Dr. Mohammad R. Rakhshandehroo, DeVry University

APPROVED FOR THE UNIVERSITY

Thos I. Williamson 01/04/08

ABSTRACT

PATTERN TOPOLOGY FOR CONTENT MANAGEMENT SYSTEM

by Mehdi Bazargan

This thesis offers a knowledge map or stable pattern language for a content management system. The stable pattern language will be defined in terms of a system of patterns that mosaic the fundamental concepts of a content management system. The core knowledge of a content management system will be deployed by using this system of patterns. The content management system pattern language will be conceptualized and understood independent from any existing model or design to facilitate its applicability to any problem in the content management domain. The stable pattern language will be produced solely based on the nature of the software stability model, which is a new methodology to analyze and design stable systems. Using the simple and yet effective guidelines of the software stability model, this thesis identifies and describes the common aspects that all applications with the same purposes share for managing content.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my wife, Asieh Zarghami, for her care and unconditional support. I also would like to thank my parents and Asieh's parents for all of their support while I was pursuing this research. Their spiritual support during the time that I was active on this extensive work gave me energy and the resolve to carry this work through.

My special gratitude goes to my advisor, Mohamed E. Fayad, who guided me through this work with his wisdom and prudence, encouraged me to proceed ahead, and supported me throughout this process. The concept of software stability, introduced by Dr. Fayad, has been a major theme in this research. Moreover, I thank Dr. Fayad for sharing his thoughts and ideas with his students and allowing them to use his knowledge and expertise in designing stable patterns and engines.

I especially would like to thank Dan Harkey and Mohammad R. Rakhshandehroo for their support and commitment to be members of my thesis committee. Dr. Harkey and Dr. Rakhshandehroo have greatly helped me with this research work. Moreover, I thank Shahriar Heidary for the time that he invested in proofreading this thesis report. Dr. Heidary has supported me greatly in finalizing this work.

I also would like to thank Ashka Vakil for her support in describing the Personalization and Dynamism patterns in the form of a template. I thank Yashwanth Nelpati for sharing his experience with using content management systems. Finally, I thank Srikanth Hegde for proofreading this work and helping out with the text and styles.

Table of Contents

List of Figures

List of Tables

Chapter One: Thesis Introduction

1.1. Introduction

Content Management Systems (CMS) are those sophisticated software systems meant for controlling and using digital content. While content could be manifested and represented in the form of charts, photos, textual paragraphs, or audio, the tasks of managing content would be quite similar for all: deleting, accessing, versioning, editing, adding, and even comparing. Managing content is neither a luxury nor a convenience in the modern age; rather, it is a necessity for daily business. Indeed, content management systems like email service mangers have subtly and effortlessly become integrated in our daily careers.

The Software Stability Model (SSM) is an emerging and new knowledge-based software development paradigm based on the idea of stability and pioneered by Mohamed E. Fayad through his empirical work of Introduction to Software Stability, How to Deal with Software Stability, Accomplishing Software Stability, and A Pattern Language for Building Stable Analysis Patterns. Further works on stability such as A Foundation for Building Stable Analysis Patterns and Building Stable Analysis Patterns Using Software Stability have been accomplished by Haitham Hamza.

SSM defines very effective methodologies for software analysis, design, implementation, and testing. The software stability model injects and infuses the groundbreaking idea of stability into the software development process, thereby, offering

stable and domain independent software engines applicable to as many relevant applications as desired.

Stable Software Patterns (SSP) are design patterns that are crafted based on the software stability model in an effective mode to represent stable concepts. Linking stable software patterns would enable and empower developers to add and integrate an unlimited number of concepts in their software engines and would allow them to develop various applications per demand.

Whether it is about accessing a file, editing a movie clip, or publishing a paragraph, all content management software available today share a set of common logic, as they all serve one specific purpose: managing content effectively. Tools designed and crafted through traditional software development methodology interweave and integrate problem concepts into application logic layer. There is no clear line of separation between an application concept and a core problem concept. In that traditional model, a software core has no or very little capacity to be extended to applications that share the same or very similar concepts but are used for different purposes. Furthermore, any change or modification to design specifications of a software application could result in unpredictably large changes in the application. This thesis work offers an effective solution to solve these problems by designing a content management engine using the software stability model so that the engine could be used in a myriad of applications without the need to change the software engine.

This research leverages its strength from the current and previous work on the software stability model and techniques and intends to create a unified content

management engine with a knowledge layer completely separate from the rest of the application. This would make the engine flexible enough to be applied to many applications of content management. Using the principles of SSM, the core knowledge of a software application will be separated from the rest of the application logic. The software stability model specifies how to use Enduring Business Themes (EBT) and Business Objects (BO) in various stages of software development to generate the core application knowledge and separate it from application-specific logic. EBTs are the enduring goals, fully stable concepts over time that are not changing across different domains. BOs are capabilities that empower the EBTs and are internally stable and externally adaptable. BOs capture and represent application's business logic and are adaptable to different applications. Beside the EBTs and BOs that form a software engine, SSM considers another class of components in design: Industrial Objects (IO). IOs are not stable; they represent application-specific modules and techniques. It is possible to hook IOs only to other IOs or BOs, whereas BOs connect only to EBTs.

First, in this chapter, the problem which this thesis attempts to address is presented in depth. Then, it specifies a solution followed by an overview of the methodology used. Next, the unified content management engine is overviewed, and the methodology to carry out this research is discussed in its entirety. The final sections of this chapter are the conclusion and references.

1.2. Problem

With the enormous volume of exchanged messages, published blogs, digital music, videos, articles, and data generated in daily business, managing the available

content has turned into a major operational concern for businesses, governmental agencies, research centers, developers, and even for ordinary enthusiasts. The problems and pitfalls with the existing content management systems are so vast and diverse and their application domains are so wide that many vendors have emerged to supply their solutions to only a tiny subset of the existing problem.

There are many different software applications for accessing and managing different contents. As stated in Gilbane's Report, "today, there is not a single, well-defined content management application that has the usual 3-6 major vendors associated with it" ("What Is Content Management?" 8). The result, therefore, is a myriad of applications and various vendors for expensive, over-featured, and incompatible software that often overlap in their functionality and specifications. Moreover, no existing content management software is flexible and adaptable enough to extend its functionality and capability to integrate into any user-defined system configuration.

Table 1.1 indicates how some of the most popular and widely used existing content management systems differ widely and how narrow their support is for various items such as supported databases.

Table 1.1

Vividness and Restrictions in Commonly Used CMSs

	Joomla	**Drupal**	**Plone**	**Bitrix**	**Flash**	**Java based**	**.Net based**
Language platform	PHP	PHP	Phython	PHP	Action script	Java	.Net
Database	Mysql	Mysql	Zope	Mysql, Oracle, MS SQL	Stores in XML files only	Mysql	Mysql
License	CNU GPL	CNU GPL	CNU GPL	License on purchase	350 Euro	based on features	based on features
Application Server	PHP	PHP	Python	PHP	Action script	-	-
Web Server	Apache	Apache, IIS	Apache, IIS, Zope	Apache, IIS	Apache	Apache	IIS
Workflow engine	No	Limited	Yes	Yes	No	Yes	Yes
FTP support	Needs plug-in	Limited	Yes	Yes	No	Yes	Yes
CGI-Mode	No	Yes	No	Yes	No	Yes	Yes

Source: CMS Matrix.2007. The Compare Stuff Network. 15 August 2007 <http://www.cmsmatrix.org>

The list of major problems in existing content management systems is discussed below:

• **Difficult to adapt and integrate:** Many organizational efforts to adapt a CMS into a business fail because, according to the Gilbane's Report, existing content management systems cannot be tuned to each organization's process flow and will not buy customers' acceptance ("What Is Content Management?").

• **Limited extensibility:** It is quite difficult and tedious to add a new component to a CMS. Although some content management systems support add-on mechanisms for extending systems' features and capabilities and provide a framework for add-on development. However, such mechanisms always fail in extending the systems in terms of infrastructure. For instance, a feature for secure logins could be added to a content management system; however, adding support for a new database is not feasible unless the CMS is partially or fully upgraded.

• **Inflexible:** Existing content management systems are not flexible enough to allow users to carry out their diverse tasks as they would desire. They also do not allow users to easily become adjusted to the emerging trend of technological or development environment changes. This would be quite costly for businesses that are dependent on new technologies.

• **Short lifespan:** Due to their inflexibility to respond to technological changes, existing content management systems have very short life spans.

• **Low return on investment:** Many existing content management systems offer features that remain unused or unexplored either because they are not flexible to be adapted to business processes or there are no needs for the features. Availability of such unused features would increase the cost of the software and would eventually reduce customers' returns on their investments.

• **Difficult to integrate deployed systems across different domains**: Due to the myriad of customers' needs and inflexibilities in existing content management systems, multiple content management systems cannot be used to meet the needs of large organizations.

1.3. Solution

This thesis proposes a brand new design for a CMS engine using the software stability model to solve the problems listed in Section 1.3. The proposed CMS engine could be used as a core software engine in any content management system software. Most of the work in this report specially focuses on proposing a design for a stable content management engine rather than trying to mosaic various techniques and algorithms to cover the problems stated.

In this thesis, the proposed CMS engine fully follows the principles of SSM and captures the core knowledge for any CMS. To construct an effective CMS software application around this engine, various IOs of choice could be hooked to BOs of the proposed engine. The following benefits are directly resulted from using the proposed CMS engine:

- **Easy to adapt:** The CMS software applications that are constructed using the proposed engine could easily be adapted to organizational needs and process flows by simply using compatible IOs and configuring the software application for the customers' settings as per need.
- **Fully extensible:** It is quite easy to add a new component to the CMS applications which use the proposed engine. Adding a new component would simply involve hooking the component to the engine as an IO. The engine will readily be equipped with all required BOs.
- **Flexible:** The CMS applications created based on the proposed engine of this thesis would be quite flexible and straightforward in keeping up with technological

advancements or changes. This is due to the fact that technologies are treated as IOs in this research, and users could develop or add their own IOs in any desired shape or format.

• **Long life span:** Due to their flexibility to respond to technological changes, those content management systems which are developed using the proposed core would have long life spans.

• **Greater return on investment**: All features of the proposed engine and the IOs which are not needed by customers can be deactivated to reduce the cost of the software. This action would increase the return on investment for customers.

• **Easy to integrate systems deployed across different domains**: The CMS applications that are developed using the proposed engine could be used for various domains or easily configured and scaled up to cover various organizational needs.

Other benefits associated with the proposed solution of this thesis are:

• **Efficient application development:** Due to code reusability and an object-oriented approach, new software application development around the proposed engine will be faster and more efficient.

• **More trustable application:** Due to extensive code reuse, many software modules of the proposed engine are exercised within various applications. This would make error detection faster, and it would exercise the engine effectively.

1.4. SSM Overview

The Software stability model introduces a unique layered concept to represent and institutionalize stability in software. The inner most layer of the three-layer concept is the EBT layer. EBT conserves the essence of the goals for which the engine is designed for. The EBT layer is internally and externally stable and has no connection with objects that represent non-conceptual entities. The middle layer surrounding the EBT is the BO layer. BOs represent the capabilities needed to satisfy the goals captured in the EBT layer. BOs are internally stable and externally adaptable; the internal state of any BO object is changeable while it is highly adaptable externally. A stable engine is created from BO and EBT elements. Finally, IO elements make the last layer around the BOs. IO objects represent physical elements in design that are changeable. To summarize, the following questions and answers explain the three layers in SSM:

- **What are the EBTs?** All goals, for which an engine is being created for, are captured in the EBT layer. All applications with the same goals can use the engine whose EBT layer captures those goals. Since EBTs are internally and externally stable and unchangeable, they cannot represent physical objects. This is due to the fact that all physical objects could be changed across applications.

- **What are the BOs?** The capabilities that enable and streamline a stable engine to pursue the goals defined in the EBT layer are the BOs. The union of the EBT and the BO layer defines an engine.

• **What are the IOs?** IO elements represent tangible and changeable objects in design. New applications could be constructed, when different objects are hooked to BO elements of an engine. Depending on which capabilities are active and which IOs are used, the applications could then be easily modified.

Figure 1.1 shows the three layers of design components as suggested by the software stability model. These layers include EBT, BO, and IO elements.

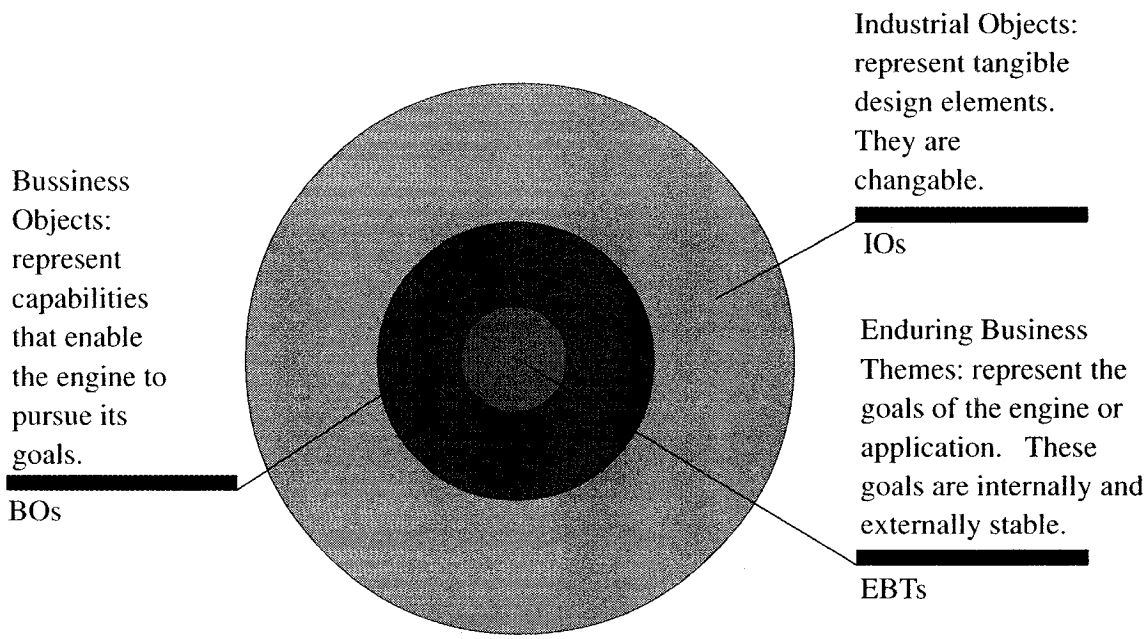

Figure 1.1. The three layers of SSM components.

Reference could be made to Accomplishing Software Stability by Fayad or Building Stable Analysis Patterns Using Software Stability by Hamza and Fayad for more information on EBT, BO, and IO layers.

1.5. CMS Engine Overview

The Digital Revolution marks the beginning of 1990 to the present by "describing the effects of the rapid drop in cost and rapid expansion of power of digital devices," as stated in Wikipedia ("Information Age"). Technical breakthroughs and discoveries have given birth to great revolutions in digital communications and information exchange during this eventful and thought provoking period. For digital communication systems, available information is considered as digital contents that could be manifested in various shapes and forms such as digital files, voices, images, and texts. Given the importance of having the latest information available in today's businesses and given the high volume of modern information exchange, it is crucial to use effective content management systems.

With so much diversity and changes in the format, use model, and volume of contents, developing a solid, adaptable, and configurable content management system has shown to be a unique and special challenge. As stated previously in the chapter's problem statement, all existing content management systems only support limited interfaces. Such systems are designed specifically for certain applications and cannot be applied to any application of managing content. Visiting major content management vendors' websites can show and reveal how such systems are separated by domain: education, commerce, finance, government, and sales. For enterprise content management systems, Webdot could be referenced for wide domain coverage ("Enterprise"). Allofe Solutions targets corporate, government, and healthcare domains ("Corporate"). Educational domains in terms of eLearning and applicant tracking are

also covered by Allofe Solutions ("Applicant Tracking"). Although some vendors offer a fair measure of personalization and flexibility in their CMS workflows, the personalization or configuration capabilities which they offer are truly limited. However, users' basic needs for personalization are always limitless; hence, they must be capable of having their own personalization modules.

The CMS engine that is proposed in this thesis will be fully adaptable, changeable, and configurable by separating the engine's goals and business logic which are common to all CMS applications from its techniques and components which are application-specific. Figure 1.2 represents the flexibility of this study's proposed CMS engine.

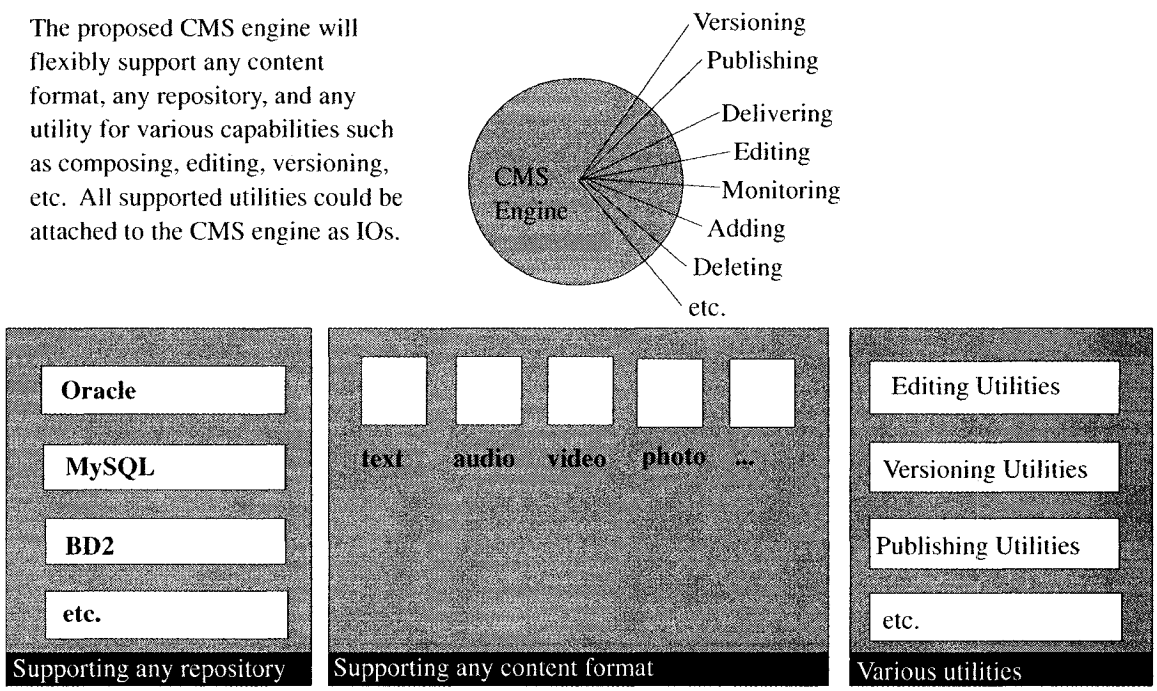

Figure 1.2. Representation of flexibilities with the proposed CMS engine.

The CMS engine could receive the data from a myriad of sources and in different formats. The content could then be customized for presentation in any desired format, stored in any type of database, or it could be delivered to any reachable destination. More importantly, the crafted engine could be used across any domain of content management: education, finance, sales, government, and many other fields.

1.6. Research Methodology

This research study has been carried out using the software stability model's techniques to achieve its goals in designing a stable, configurable, and adaptable content management engine. In this research study, as stated and described before, the application knowledge, which is defined in terms of EBT and BO elements, is fully separated from all application dependent modules. The following phases and steps are the major strides in achieving the final content management engine design as a system of patterns:

1. Analysis phase: The main goal in this phase is to develop analysis patterns, and this involves the following important tasks:

• Identifying the goals: This study involves intuitive and empirical attempts to detect and identify all possible goals for a content management system. This is a search for the true goals of the system. When similar goals exist, the challenge is to identify the most apt among all. The identified and selected goals will make the EBT layer. It is critical to identify EBT elements correctly; otherwise, the engine's stability would be jeopardized.

• Developing analysis patterns: The last important task in analysis phase is to create analysis patterns based on the identified EBTs.

2. Design phase: The main aim here is to develop design patterns in this phase which involves the following steps:

• Identifying capabilities: The next step in design is to identify the capabilities needed to drive the identified goals. These capabilities are the BO elements that will be connected to the EBT elements in the next step.

• Developing design patterns: The critical task in this phase is to develop design patterns for the identified BOs.

• Developing knowledge map: This would involve using the analysis and design patterns identified in the previous steps to create the knowledge map for a content management system. Knowledge Map (KM) is a stable knowledge core that maps goals and capabilities in design and analysis patterns to form an engine.

• Developing framework: This step involves using the knowledge map developed in the previous step and the patterns needed by CMS application specifications to generate an engine framework.

Another important task that shall be accomplished in this phase is verifying that the collected EBTs, BOs, and the constructed patterns in previous steps are indeed the necessary and sufficient engine blocks. In case a scenario missed a design component, the source of the problem in design or analysis patterns should be identified heuristically and the previous steps shall be repeated.

3. Development and implementation phase: In this phase, the application development is performed by the following steps:

• Define the IO elements: First, prepare the specifications for the application of interest. Then, use the specifications to identify the IO in the context of SSM. IOs are the application-specific modules of the software corresponding to the engine interfaces.

• Implement: Implement the code for all the identified analysis patterns, design patterns, and all required IOs.

• Test and validate: Validate and verify the functionality correctness and aptness of the code using applicable test cases.

4. Integration phase: In this phase, the developed patterns are connected to form an engine. Then the engine is connected to application IOs. Later, the new system is compiled and built.

5. Configuration phase: Based on the application documents, the developed application is tested and queried in certain scenarios and the feedback is generated later. Based on the nature of the feedback, the software may require further adjustments. Such adjustments could be applied in the analysis and design phases.

6. Operation phase: This step involves installing, setting up, and using the developed software in its application domain based on some guides and related software documents. In this phase, the application shall be fully documented for users and be ready for use.

7. Maintenance phase: There would be no maintenance hassles for the software applications that are crafted using the software stability model as the application would be fully stable over time. However, some software maintenance might be required in

case some software bugs or problems are reported. In case of technological changes and modifications in application specifications, maintenance will be as simple as attaching a new IO to the engine or detaching an existing one. All reported bugs must be documented for possible later analysis. Figure 1.3 represents various stages of software development life cycle based for the software stability model. These stages are categorized into three different groups for simplicity.

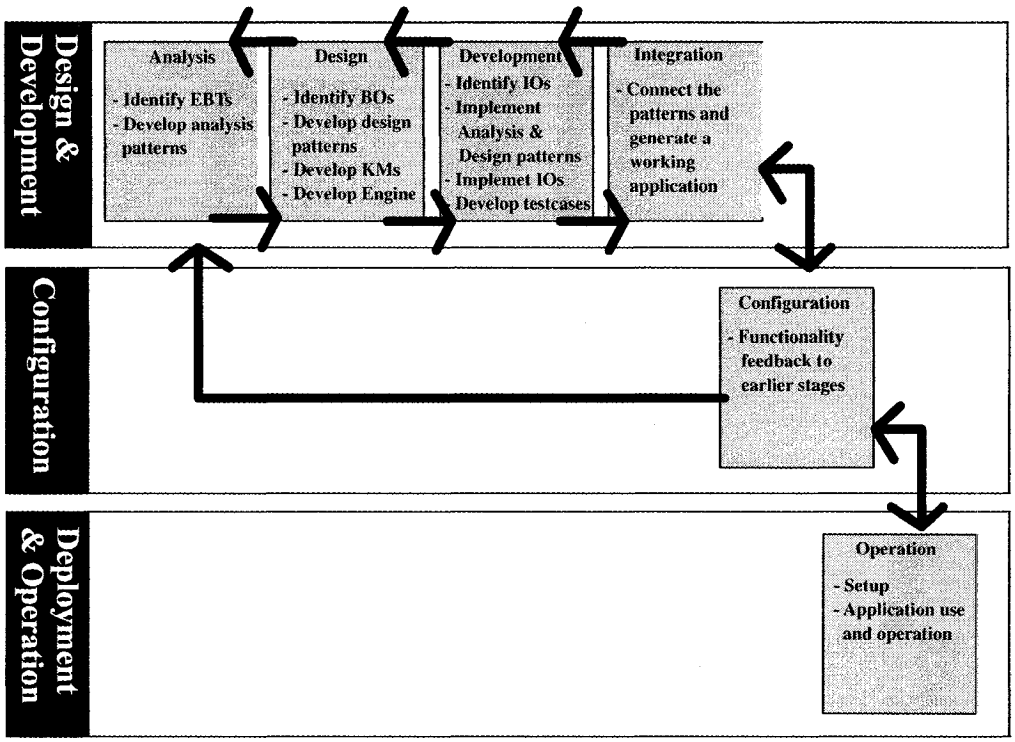

Figure 1.3. Various stages of the stable software development life-cycle.

1.7. How This Research Was Conducted

To carry out this extensive research, Mehdi Bazargan, who is the author of this report, attended a series of meetings with the software stability group, which was lead and advised by Mohammed E. Fayad. During these meetings, many topics on SSM,

software development practices, and past, present, and future research work were discussed and presented. The meetings were regularly held on a weekly basis and in some cases more frequently. Furthermore, this thesis involved researching available content management systems and studying many related articles to identify the pitfalls of some major content management systems. After identifying the pitfalls, several methods to resolve such problems were explored.

Furthermore, this unique work was accompanied by learning and using the software stability model and design. It also involved some effort to identify the goals and capabilities for a content management system. Furthermore, this work involved developing some stable software patterns and creating a knowledge map from them. Moreover, many unique scenarios were explored and visited to validate the offered design. This research also involved enhancement and supplementation of an available tool used in stable software development and applying the enhanced tool to applications developed based on SSM. Finally, much time and effort was spent to compose and review this thesis report.

1.8. Thesis Contributions

This work makes several contributions. This thesis provides the basic construct that could be used for future engine development. Moreover, this work offers an effective CMS engine that could be used in any CMS application. The following items are contributions made by this thesis:

• The comparative study performed for this thesis on available content management systems identifies the pitfalls that negatively impact CMS application development, CMS application usage, and its maintenance. Such findings open the way for further inquiry into the problems and pave the way for finding some solutions for the identified issues.

• In this research project, some goals and capabilities for a content management system engine have been identified using the SSM approach. The goals and capabilities are also known as EBTs and BOs respectively. Using the identified EBTs and BOs, some stable software patterns were developed. Moreover, in some cases, existing software patterns were used and plugged into various scenarios for verification and validation. The developed stable software patterns are valuable commodities. Such stable patterns could be used for engines that share common goals with the stable patterns. Therefore, the stable software patterns could be added to a bank of available stable software patterns for further software and design reuse.

• The next major achievement of this thesis is the construction of a stable content management engine from some improvised or available stable software patterns. This engine is the core achievement of this thesis. It resolves all problems addressed in Section 1.3, and it could be used in various CMS applications.

• Another important contribution of this work is the implementation of the mentioned content management engine which was designed using some stable patterns. In this work, the engine is documented and implemented. With such tangible and significant contributions, this thesis prepares the ground for further expansion of the work and

creation of a CMS application from the engine which resolves the problems of many existing content management systems.

1.9. Thesis Layout

This thesis report consists of seven chapters and four appendices. In addition, there is a separate bibliography at the end. The chapters and appendices cover the following content:

- **Chapter One: Introduction**

This introductory chapter introduces the topic of this thesis and presents a list of major problems and pitfalls associated with the modern content management systems. It also introduces the concept of software stability. Finally, a solution to the stated problem is overviewed in this chapter.

- **Chapter Two: Comparative Studies**

Chapter Two involves a detailed study of various content management systems that exist in today's market. It compares the capabilities of six major content management tools that have the greatest number of users. This comparative study is meant to identify the shortcomings and strengths of each studied tool.

- **Chapter Three: UCME's Basic Building Blocks**

Chapter Three introduces the basic building blocks for a content management engine. It also describes some of the major goals and capabilities involved in designing a stable content management engine.

- **Chapter Four: The Unified Content Management Engine**

Chapter Four introduces a CMS engine as a system of stable patterns. It suggests a new classification based on the SSM guidelines and explores some SSM classifications.

- **Chapter Five: Case Study**

In Chapter Five, the introduced CMS engine will be used to build a CMS application. This is meant to explain how a CMS software application could be constructed from the proposed CMS engine.

- **Chapter Six: Analysis & Major Findings**

Chapter Six analyzes the major achievements of this research work and explores all associated findings.

- **Chapter Seven: Future Works & Conclusions**

Chapter Seven discusses the ways in which this work could be expanded, diversified, and continued in future.

- **Appendix A: Personalization Stable Analysis Pattern**

Appendix A documents the Personalization analysis pattern as referenced in this thesis report.

- **Appendix B: Dynamism Stable Analysis Pattern**

Appendix B documents the Dynamism analysis pattern as referenced in this thesis report.

- **Appendix C: Sample Code**

Appendix C contains sections of code used in the implementation of the Personalization analysis pattern.

- **Appendix D: Demonstration**

Appendix D contains some material and snapshots for demonstration of the implemented Personalization analysis pattern.

Chapter Two: Comparative Studies

2.1. Introduction

Content management systems are software systems exclusively meant for controlling and managing digital content. Content management systems range from simple to complex with costs ranging from free to tens of thousands of dollars, and could be deployed over various domains. Content management systems have wide applications in healthcare, commerce, sales, engineering, publications, government, education, and all other fields in which content plays a central role in business activities. Content management systems help to manage a high volume of data stored in various repositories and vivid formats using different technologies.

Digital content could be any data or information that is digitally stored anywhere such as in remote databases or locally as files. Contents could be of various types such as photos, audio, videos, texts, and meta-data. Contents could also take various formats such as MP3, MIDI, and WMA. Photos could be in other formats such as JPEG, BMP, or TIFF. In this thesis report, a digital content is implied whenever there is a reference to content.

Content management systems are usually equipped with a myriad of capabilities to manage a high volume of data of various types and formats. Editing, formatting, deleting, creating, re-shaping, delivering, publishing, gathering, authenticating, and deploying are only a few capabilities that the modern content management systems should be able support. In fact, many difficulties arise when content management

systems fail to support capabilities that are required by users. For example, to authenticate users before granting them access to some protected contents, some protocols such as LDAP and SSL could be used. A flexible content management system should be able to support all capabilities that customers need in order to gain the customers' acceptance.

In Chapter Two, the main intention is to introduce some selected content management systems and compare and evaluate them to study the advantages and disadvantages of those selected systems. First, six major content management systems are selected and overviewed. Then, a set of criteria for comparison purposes are introduced and discussed. Next, the criteria are used to compare the selected systems. Then, an analysis of the capabilities and strengths of the selected systems is presented. Finally, a conclusion section recaptures all topics discussed in this chapter.

2.2. Major Content Management Environments

In Section 2.2, six different content management systems will be presented and discussed. Since this chapter aims to compare various content management systems in terms of their capabilities, strengths, and shortcomings, the systems being compared shall have similarities in their functionalities and purposes. Moreover, since the applicability domains for content management systems are vast and numerous, this report considers the web domain as a specific and special domain of applicability, and it will compare selected content management systems in that domain. Focusing on a specific domain will not impact the results of this comparative study as design methodologies for developing

content management systems remain similar across different domains. For that, all usability advantages and disadvantages associated with such design methodologies could still be captured by limiting the domain of this study to the web domain.

As established above, in this comparative study, the focus would be narrowly on the web domain. The web domain is certainly the most applicable domain for studying content management systems. This is due to the fact that all contents generated in various domains such as education, finance, marketing, and government will still need to be displayed and accessed through various means. Publicizing these contents through the web is the most popular method in our time.

The following content management systems are chosen for this comparative study because they are vastly used, designed differently, and all have different audiences:

1. Joomla CMS: Joomla is a relatively popular content management system. It recently branched off from Mambo CMS, and it shares some known history with Mambo. This well-featured content management system supports many useful and beneficial web components for beginners and small businesses.

Joomla CMS uses some add-on components to further extend its capabilities. Some add-on modules are readily available for Joomla free of charge under a General Public License (GPL). Some of these components are Joomla's dynamic form builders, business or organizational directories, document management, image and multi-media galleries, e-commerce and shopping cart engines, email newsletters, and some blogging software as mentioned in the official Joomla website.

Joomla also offers an application framework for new software component development that would allow Joomla users to add some new components to their Joomla installation. Joomla lists some of these modules as "integrated e-commerce systems, inventory control systems, data reporting tools, custom product catalogs, complex business directories, reservation systems, communication tools, application bridges or any kind of application to suit a unique need" ("What Is Joomla?").

Although Joomla is rich in offering various gadgets, it simply fails to support various infrastructure components. Some of such infrastructure could be as important as the Oracle database management system. Certainly, organizations that run mission critical applications and larger businesses that could afford employing durable applications such as Oracle for their business advantages will not be able to use this CMS. In other words, Joomla is flexible for application development on some specific platforms, but it is quite rigid and inflexible at the system levels.

Despite of its free of charge policy, Joomla is still under the GPL license and has restrictions that some commercial businesses may not be willing to embrace.

2. Plone CMS: Plone is a web content management system that in terms of constructs and architecture is almost similar to Joomla. Plone is free of charge, and it offers several add-on modules to extend the capabilities of the CMS. In addition, Plone offers many add-on security modules that set the CMS on a higher ground compared to other content management systems of the same class such as Joomla.

Despite the extensibility advantages in terms of adding extra component to the system, Plone offers support for infrastructures such as Oracle database management system. Plone is based on Zope and supports selected relational databases: MySQL, ZSQL, Postgre SQL, and MS SQL. As mentioned in Wikipedia, Zope is an object-oriented, open source application server based on the Python programming language ("Zope"). This CMS supports Microsoft Windows, Linux, and Mac OS but fails to support UNIX.

Plone also offers a development framework for extending the CMS with new add-on modules. Plone website indicates that Archetypes are the framework for developing new content types for any Plone project ("What Is Archetypes"). This framework has advantages over simple inheritance and sub-classing techniques. The following are some of the advantages that ease developers' jobs by following up with Archetypes:

- automatically generates forms and views;
- provides a library of stock field types, form widgets, and field validators;
- allows defining custom fields, widgets, and validators;
- automates transformations of rich content;
- a built-in reference engine that gives the ability to link two objects together with a relation; such a "link" from a given object to another one is a Python object called a reference. ("What Is Archetypes," Plone)

3. SharePoint Server (.NET Based CMS): Microsoft offers an Enterprise Content Management (ECM) system meant for document management, records management, web content management, and forms solution as indicated by OfficeOnline ("Microsoft Enterprise Content Management"). The web content management has many components readily available for use as part of the software package. There exists no add-on business

with this Microsoft product; all are set available. Users may choose to create their customized templates and employ them in their development activities. The MS SharePoint is quite strong on deployment, especially for multi-tier infrastructures: internet, intranet, and extranet sites. It also provides attractive capabilities for secure computing and computational performance. However, based on the available information from CMSMatrix, MS SharePoint is not very strong in providing capabilities for e-commerce and online commercial transactions.

MS SQL is the supported database server for MS SharePoint. MS SharePoint supports C#, ASP.NET, and ASP languages. Moreover, based on the available information from CMSMatrix, applications written in almost any language can be effectively integrated with the CMS. Based on the available information from OfficeOnline, MS SharePoint is offered on commercial licenses and the price is not quoted for the SharePoint Server. Moreover, MS SharePoint's licenses are only available through volume licensing ("2007 Office"). The estimated price for obtaining annual licenses of MS SharePoint Server 2007 for Internet Sites is about 41,000 U.S. dollars based on the OfficeOnline price list ("Microsoft Office").

4. SCMS Flash (Flash Based CMS): This is a content management system which is exclusively designed for Flash web pages. Based on the SCMS Flash Content Management official website, SCMS Flash supports users' needs for managing pictures, texts, animations, and other special flash components. This CMS is not as efficient when it is compared with the other selected content management systems of this study. SCMS Flash does not support e–commerce, is not extensible with any add-on module, and is not

supportive of various applications based on the information available from CMSMatrix. This CMS is meant to support flash web pages and is very focused on providing capabilities for editing visual components only.

SCMS Flash recognizes MySQL as the only supported database and ActionScript as the only supported programming language. SCMS Flash could only use Apache web server and costs about 350 Euros by the CMMatrix's price list.

5. Apache Lenya (Java Based CMS): This content management system supports various databases and strong programming languages such as Java to bring flexibility for CMS integration in workflows. However, Apache Lenya does not support e-commerce. Apache Lenya it is not supportive of built-in applications as mentioned in CMSMatrix. This content management system uses Apache Cocoon as the framework of development based on the information available from Apache Lenya. As stated in The Apache Cocoon Project, "Apache Cocoon is a web development framework built around the concepts of component-based web development and separation of concerns, ensuring that people can interact and collaborate on a project without stepping on each other toes."

Apache Lenya supports UNIX and Microsoft Windows operating systems. Based on the available information from The Apache Cocoon Project, this CMS has a good record of security and performance with support of various performance tuning capabilities such as advanced caching, load balancing, page cashing, and static content export. In terms of security, it is stated in the Apache Lenya's official website that the CMS supports SSL, LDAP, and various other access control mechanisms ("Apache Lenya - Open Source Content Management").

Apache Lenya's official website states that this CMS is free of charge, open source, and its usage is governed by Apache Licensing ("Apache Lenya – License").

6. Drupal CMS: The Drupal's official website states that "Drupal is a free software package that allows an individual or a community of users to easily publish, manage, and organize a wide variety of content on a website" ("About Drupal"). This content management system application supports some add-on capabilities to extend the application free of charge with any available module of interest. Based on the available information from CMSMatrix, Drupal only supports Postgre and MySQL database servers and it accepts PHP as its sole supported programming language. Therefore, despite the fact that Drupal supports many add-on and built-in applications, it simply fails to flexibly support different infrastructures.

Drupal also supports Apache and IIS web servers. Its usage is governed under GPL terms of use, and it is free of charge to obtain the application software.

2.3. Comparative Criteria

The main criteria and measures considered in this research for comparing the selected content management systems are listed and discussed in Section 2.3. The comparative criteria are chosen to be the most determining key capabilities that impact the usability of a content management system application.

1. Applicability: This special criterion represents the range of CMS capabilities. Some content management systems only manage simple and static websites. Others, in a similar class, could even support secure remote access for full site development,

managing financial transactions, tracking legal violations, and a host of other relevant capabilities that enable users to effectively manage a more sophisticated organizational website. For example, SCMS Flash is a CMS with limited static capability. On the other hand, based on the available information from the Bitrix Site Manager, Bitrix CMS is a sophisticated content management system for web development with extended and enhanced capabilities ("Features and Modules").

The more capable content management systems are not necessarily the most superior in terms of functionality. Superiority of a CMS should be rather measured based on satisfaction of users' needs with regards to the CMS capability. This is particularly true because the content management systems that are equipped with more capabilities tend to be more complex, less flexible, and even tend to require more maintenance. However, not speaking of superiority, this criterion is about availability of features that are applicable to users' needs for effectively managing web contents. Such capabilities may range from simple editing to multi-tier site publishing.

2. Extensibility: This criterion refers to the possibility of extending content management systems' capabilities in various dimensions such as infrastructure and technology. For example, a CMS that does not support financial transactions management could be extended to do so. Similarly, extending a CMS to interpret and work with new technologies such as new protocols or programming languages is an example for technological extensions. The idea of extensibility implies that a CMS should be extendable in various dimensions without the need to change the core software.

Currently this is performed through the use of plug-ins, which are modules meant to add new capabilities to an application.

3. Cost and license: One of the determining factors in ranking a content management system is the cost of the system. A system's price is directly related to the count and types of available features on the system. The smart systems would only support features that are necessary for their successful functionalities. However, such systems must remain extensible to support possible future needs for adding in capabilities. In addition, with regards to the cost, the type of licenses granted is of great and critical importance. For example, a free content management system application with the General Public License (GPL) will not be useful for businesses because the GPL license is restrictive.

4. Security: A major criterion in comparing content management systems is security. A CMS with non-established record of trust and fidelity would certainly not receive acceptance of users whose businesses rely on data privacy. The following capabilities in secure computing are the standard capabilities that all secure content management systems must be equipped with:

- **Audit trail and session management:** This refers to a system's capacity in creating logs for tracking various users' activities.

- **Cryptography:** This involves encrypting communications between clients and servers as per need.

• **Standard authentication and authorization services:** This refers to supporting current business practices for authentication and access authorization. Kerberos and Lightweight Directory Access Protocol (LDAP) are examples of such services.

• **Login history:** This involves logging the history of past users' access to certain secured resources.

5. Flexibility: This criterion refers to a content management system's capacities in flexibly carrying out various users' missions and tasks.

Flexibility of a content management system could be measured by degree of available flexibilities in its components. For example, a scripting language such as PHP is considered a flexible language for web development. Therefore, a content management system whose native capabilities are technically more lenient is also considered to have some degrees of flexibility. In this regard, availability of the following features or capabilities could increase flexibility of a content management system: CGI-mode, support for content reuse, extensible profiles, metadata support, multi-lingual content support, multi-tier site deployment support, URL rewriting support, wiki awareness, and interface localization.

6. Performance: This criterion mainly refers to a content management system's throughput. A satisfactory performance is achieved with the least number of failures, maximum number of responses, and the time to process a request. There are a series of capabilities that increase the chances for better performance:

• **Advance caching:** This is a service for quick content retrieval.

• **Load balancing:** This involves balancing the incoming requests traffic across the active servers to avoid stretching a particular system's resources.

• **Database replication:** As noted in Wikipedia, this refers to "the process of sharing information, so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability and fault tolerance" ("Replication").

• **Page caching:** This involves caching users' requests before writing them back to a disk at a later time when the requests are processed.

7. Deployment: This criterion is meant to measure how easily a content management system could be deployed. Certainly, the size and complexity of a CMS impacts its immediate deployment. The larger and more complex content management systems are the more difficult and strenuous they are to deploy compared to smaller systems. After all, larger systems will require additional techniques to maintain their defined performance criteria.

Among the content management systems that are similar in size and complexity, those systems that could be deployed in a simpler manner have higher advantages compared to the rest.

8. Adaptability: This criterion refers to a content management system's capacity to be used in various fields. This study only considers the content management systems that are used for web contents. However, there are content management systems for various other domains beside the web domain. When an organization requires a content management system for its various activities such as research, development, or

marketing, the solution should be a unified and adaptable system that could cover various needs across the organization. In other words, one unified content management system should be deployed and used across different organizational domains to manage various types of content.

9. Stability: This refers to a content management system's capacity in coping with changes to the system's software application specifications. In other words, a stable content management system would have capacities in tolerating a substantial switch to its deployment environment and the system's specifications. The term *tolerating* means that the same content management system could be used in a new environment without the need to update the CMS application's code or design.

10. Accessibility: This refers to CMS capabilities in making contents accessible in various formats, through various channels, and across various fields. For example, contents could be set accessible through site login or Native File System (NFS) in various formats such as PDF, HTML, XML, and MS Word across an organization's finance, marketing, and engineering departments.

11. Integration: This refers to integration of various content management systems that are deployed across different departments of a wide organization. For example, a corporation's Research and Development (R&D) unit uses a specific CMS from a vendor, and the same corporation's Sales department uses another CMS from a different vendor. This criterion is meant to study how the content available in the R&D unit of the corporation could be used in Sales department by integrating various content

management systems used in each department into one unified CMS for better exchange of the available contents.

2.4. Comparative Study

In Section 2.2 several content management systems were selected for a comparative study. In Section 2.3, a set of criteria and parameters for comparison purposes were established and described. In Section 2.4, the discussion is continued by comparing the selected content management systems based on the established criteria. Only one criterion is picked and used at a time when discussing and comparing the selected content management systems. Although some criteria such as extensibility and flexibility are closely related, this study would be merely focused on the mentioned criteria and will refrain from following the string of relationships into different realms of discussion.

1. Applicability: As established in the paragraphs of Section 2.3, applicability mainly refers to the extent that a CMS is usable for a certain purpose. Based on this definition, the various capabilities for the selected six content management systems are presented in Table 2.1 based on the information available at CMSMatrix.

Table 2.1

Various CMS Capabilities for Applications of Web

		Content Management Systems					
Built-in Applications & Other Capabilities		**Apache Lenya**	**Drupal**	**Joomla**	**Plone**	**SCMS Flash**	**Share Point Server**
	Blog	Yes	Yes	Yes	Yes	No	No
	Chat	No	Free Add-On	Free Add-On	Free Add-On	No	Costs Extra
	Classifieds	No	Free Add-On	Free Add-On	Free Add-On	No	Yes
	Contact Management	No	Free Add-On	Yes	Free Add-On	No	No
	Events Calendar	No	Free Add-On	Free Add-On	Yes	No	Yes
	FAQ Management	No	Yes	Yes	Free Add-On	Costs Extra	Yes
	Graphs and Charts	No	No	Free Add-On	Free Add-On	Costs Extra	Yes
	Guest Book	No	Free Add-On	Free Add-On	Free Add-On	Costs Extra	Yes
	Job Postings	No	Free Add-On	Free Add-On	Free Add-On	No	Yes
	Newsletter	No	Free Add-On	Free Add-On	Free Add-On	Costs Extra	Yes
	Photo Gallery	No	Free Add-On	Free Add-On	Yes	No	Yes
	Polls	No	Yes	Yes	Free Add-On	No	Yes
	Product Management	No	Free Add-On	Yes	Yes	No	Yes
	Search Engine	Yes	Yes	Yes	Yes	No	Yes
	Site Map	No	Free Add-On	Free Add-On	Yes	Costs Extra	Yes

Table 2.1 continued...

	Stock Quotes		Free Add-On	No	Free Add-On	No	Yes
	Surveys	No	Free Add-On	Free Add-On	Free Add-On	No	Yes
	Advertising Management	No	Free Add-On	Yes	Free Add-On	Costs Extra	No
	Themes / Skins	No	Yes	Yes	Yes	No	Yes
	Web Statistics	Yes	Yes	Yes	Free Add-On	No	Yes
	Web-based Style/Template Management	No	Yes	Yes	Yes	Yes	Yes
	Inventory Management	No	Free Add-On	Free Add-On	Free Add-On	No	Limited
	Pluggable Payments	No	Free Add-On	Free Add-On	Free Add-On	No	Limited
	Pluggable Shipping	No	Free Add-On	Free Add-On	Free Add-On	No	Limited
	Pluggable Tax	No	Free Add-On	Free Add-On	Free Add-On	No	Limited
	Point of Sale	No	No	Free Add-On	Free Add-On	No	Limited
	Shopping Cart	No	Free Add-On	Free Add-On	Free Add-On	No	No
	Subscriptions	No	Free Add-On	Free Add-On	Free Add-On	No	Limited
	Wish Lists	No	Free Add-On	Free Add-On	No	No	Limited

Source: CMS Matrix.2007. The Compare Stuff Network. 15 August 2007 <http://www.cmsmatrix.org>

As presented in Table 2.1, systems like Drupal, Joomla, and Plone offer the most effective support for various applications that users could be interested in. The value

"Yes" in the column represents automatic availability of the feature, and value "Free Add-On" represents free availability of the feature as per need. There exist great interests in features such as search engine, photo gallery, advertising management, and themes. Availability of such features could increase the applicability of the CMS for various web purposes ranging from personal sites to corporate and online stores. Drupal, Joomla, and Plone provide the most number of available features. In contrast, Apache Lenya and SCMS Flash support the least number of available features and built-in applications. They offer no support for even the basic popular features such as Photo Gallery. MS SharePoint Server 2007 supports the majority of built-in applications, but it is limited in commerce. Therefore, MS SharePoint 2007 is rich in features, but it is not so applicable for commercial purposes.

2. Extensibility: This criterion is used to measure a content management system's extensibility both in terms of extensions to capabilities and extensions to infrastructures such as new database servers, protocols, and application web servers.

In the case of extensibility for application capabilities, Drupal, Joomla, and Plone offer add-on modules that enable users to add their needed capabilities to their content management systems. The three content management system applications are extensible beyond that as they provide development frameworks to extend their existing features under General Public License. It is important to note that the measure of extensibility for any content management system that provides a framework to extend its features solely depends on the power and flexibility of its framework. In other words, a CMS with a limited development framework can be extended limitedly. For example, in case of

Joomla, the add-on modules that are developed through Joomla development framework are required to use specific interfaces associated with the framework. Therefore, development of any new module is limited by the framework of development.

Apache Lenya does not provide any readily available add-on module; however, it offers Apache Cocoon framework for development under Apache license. The MS SharePoint Server 2007 and SCMS Flash do not offer add-on modules nor do they provide frameworks to extend the systems. It shall be noted that a content management system's features that enables users to create their own web templates are not considered as extensibility features.

3. Cost and licensing: In terms of cost and licensing, Joomla, Drupal, Plone, Apache Lenya, and a beta version of SCMS are free under various licenses based on the information gathered from Joomla, CMSMatrix, CMS Flash Content Management, Apache Lenya, Drupal, and Plone. However, Plone uses Zope database and application servers that are quite expensive. Based on the information made available by OfficeOnline, MS SharePoint Server for websites costs about 41,000 U.S. dollars ("Microsoft Office"). The final version of SCMS Flash costs about 350 Euros as stated in CMSMatrix. Although, some content management systems are free for software download and use, their vendors offer services that cost variably.

In terms of licensing, Joomla and Drupal are under GPL terms of use. Plone is under GNU GPL license. MS SharePoint is licensed under the commercial Microsoft terms of use. Apache Lenya could be used under Apache license, and SCMS Flash is now under SCMS commercial license based on the information gathered from Joomla,

CMSMatrix, CMS Flash Content Management, OfficeOnline, Apache Lenya, Drupal, and Plone ("Microsoft Office").

As established in the previous paragraphs, content management systems are generally inflexible and non-adaptable; hence, many features supported by enterprise systems may not be effectively used by users. In addition, the enterprise content management systems such as MS SharePoint are highly expensive. These would signal low returns on investments for customers because many features that are paid for may not be effectively used.

4. Security: The security of the six content management systems selected for this study depends entirely on the systems' supports for standard security protocols and capabilities. Table 2.2 compares such capabilities for the six selected systems.

Based on Table 2.2, Plone supports the most security features with the add-ons installed. Apache Lenya supports the second most security features with the add-ons installed, followed by MS SharePoint Server. Drupal is missing various standard security capabilities such as SSL logins and Kerberos Authentication. Worse than Drupal in security is Joomla with no support for SSL logins, Kerberos Authentication, and even Audit Trail for auditing users' accesses to any restricted content. The availability of commercial features that Joomla offers in absence of any solid security feature makes Joomla CMS more unsafe. The combination of strong but insecure features places Joomla in dangerous and potentially disastrous situation.

Table 2.2

Content Management Systems' Support for Security Factors

	Content Management Systems						
Security Factors	**Security**	**Apache Lenya**	**Drupal**	**Joomla**	**Plone**	**SCMS Flash**	**Share Point Server**
	Audit Trail	Yes	Yes	No	Yes	Costs Extra	Yes
	Captcha	No	Free Add-On	Yes	Free Add-On	No	Yes
	Email Verification	No	Yes	Yes	Yes	No	No
	Granular Privileges	Yes	Yes	No	Yes	Yes	Yes
	Kerberos Authentication	No	No	No	Free Add-On	No	Yes
	LDAP Authentication	Yes	Free Add-On	Free Add-On	Free Add-On	No	Yes
	Login History	Yes	Yes	Yes	Free Add-On	Costs Extra	Yes
	NIS Authentication	No	No	No	Free Add-On	No	No
	Pluggable Authentication	Yes	Yes	No	Yes	No	Yes
	Session Management	Yes	Yes	Yes	Free Add-On	Costs Extra	No
	SMB Authentication	Yes	No	No	Free Add-On	No	No
	SSL Compatible	Yes	Yes	No	Yes	No	Yes
	SSL Logins	Yes	No	No	Free Add-On	No	Yes
	SSL Pages	Yes	No	No	No	No	Yes
	Versioning	Yes	Yes	Yes	Yes	No	Limited

Source: CMS Matrix.2007. The Compare Stuff Network. 15 August 2007 <http://www.cmsmatrix.org>

5. Flexibility: The flexible content management systems have two different properties at the minimum. First, they can run different missions and tasks. Second, they provide flexible features to work and interface with. Users can also add, change, or disable features and capabilities of any flexible content management system. Joomla, Drupal, Apache Lenya, and Plone provide different frameworks to new features. Availability of such frameworks adds to the flexibility of a content management system. However, this does not indicate that the content management systems which offer some development frameworks are necessarily flexible. In fact, Joomla, Drupal, and Plone are solely featured through their add-on capacities. Many might be deceived at the first glance by seeing Joomla, Drupal, and Plone offering many add-on components; however, it would soon become evident that despite of such availabilities, these content management systems are quite rigid and inflexible to work with. Certainly users could develop their own add-on modules using the provided frameworks. However, not much could be done when such frameworks do not support what users really need. As for MS SharePoint Server and SCMS Flash, neither of these two content management systems provides any feasibility for a bit of change. However, unlike SCMS Flash, MS SharePoint Server offers a variety of features and capabilities to compensate for the lack of any development framework. For example, MS SharePoint Server provides the flexibility of publishing web content to one destination or multi-tier website for intranet, an extranet, and the internet. Table 2.3 shows several factors that could be used to measure a content management system's flexibility from the information available at CMSMatrix.

Table 2.3

Flexibility Factors for Various CMSs

		Content Management Systems					
Flexibility Factors	**Flexibility**	**Apache Lenya**	**Drupal**	**Joomla**	**Plone**	**SCMS Flash**	**Share Point Server**
	CGI-mode Support	No	Yes	No	No	No	No
	Content Reuse	Yes	Limited	Yes	Yes	No	Yes
	Extensible User Profiles	No	Yes	Yes	Yes	Yes	Yes
	Interface Localization	Yes	Yes	Yes	Yes	No	No
	Metadata	Yes	Yes	Yes	Yes	Yes	Yes
	Multi-lingual Content	Yes	Yes	Free Add-On	Yes	Yes	Yes
	Multi-lingual Content Integration	No	Free Add-On	Free Add-On	Yes	No	Yes
	Multi-Site Deployment	Yes	Yes	Free Add-On	Yes	No	Yes
	URL Rewriting	Yes	Yes	Yes	Yes	No	Yes

Source: CMS Matrix.2007. The Compare Stuff Network. 15 August 2007 <http://www.cmsmatrix.org>

In Table 2.3, the availability of CGI-Mode provides the flexibility of using some existing scripts to run various tasks of interest. Drupal is the only CMS that supports this feature. This is certainly a negative point for the rest of the content management systems. Content Reuse is a feature available in certain content management systems. It enables users to maintain and reuse the existing content. This would reduce the maintenance overheads of updating content. The extensible user profiles allow users to extend their

existing profiles to include other desired contents. The selected content management systems in this study, except Apache Lenya, provide extensible user profiles. The interface localization feature enables developers to localize the user interfaces of their websites. For example, developers could use some existing interfaces and localize them for a specific audience by changing the displayed language of the website or reporting the published measurements by certain standards. The metadata feature provides information about the data received from users' interactions with a server. Such metadata would aid the system to classify users' request more efficiently in order to serve them better. For example, using certain available metadata, one could receive information about the geographical location from where a request for accessing some content was made. All content management systems under this study support the metadata feature. Availability of multi-lingual content and content integration features could satisfy users' needs for multi-lingual content manipulations. The selected content management systems in this study, except SCMS Flash, support this feature. The multi-site deployment feature makes it flexible for users to choose various destinations to which they desire to publish some web content. This feature is supported by all content management systems in this comparative study except SCMS Flash. As stated in Wikipedia, URL rewriting is a feature that allows rewriting the URLs of some web contents when the contents are fetched from their sources for the advantages listed below ("Rewrite Engine"):

- Making the URLs in a website to be more usable and searchable.
- Obstructing undesired hot linking.
- Hiding implementation-specific information conveyed by URLs.

URL rewriting is an available feature in all content management systems selected for this study except in SCMS Flash.

With regards to the comparisons made above, Plone has the most number of features implemented for application flexibility.

As stated at the start of Section 2.3 and Section 2.4, the flexibility of a content management system is measured by how users could carry out their missions and tasks using the available features of the system. Current content management systems do not provide facilities and amenities to use different programming languages. For example, if an existing web project requires a Java application server, it would not be possible to use Joomla, Drupal, or Plone as none support a Java application server. Therefore, although a language such as PHP is flexible and is supported by Joomla and Drupal, these content management systems lack a number of fundamental flexibility features. Plone uses Zope application server. A similar case exists with the selected content management systems' support for various programming languages. Drupal and Joomla only support PHP. In contrast, Apache Lenya supports Java, XML, and its offshoots like XSLT, JSP, and JavaScript. MS SharePoint uses C#, ASP.NET, ASP, but CMSMatrix claims that other programming languages could be supported and integrated by MS SharePoint CMS. Plone only uses Python. None of the six selected content management systems support the widely used Perl programming language. Users cannot use low cost content management systems like Joomla, Plone, or Apache Lenya for their mixed code web application. Limited support for various programming languages is a drawback for many existing content management systems. The aforementioned cases highlight the

inflexibility problem in current content management systems. In fact, most available content management systems may not be considered flexible at all.

6. Performance: In Section 2.3, the performance criterion was introduced, and its sub-components were defined. At this stage, some performance measurements of the content management systems used in this study are compared using Table 2.4 and based on the information available at CMSMatrix.

Table 2.4

Performance Factors for Various CMSs

		Content Management Systems					
Performance Factors	**Performance**	**Apache Lenya**	**Drupal**	**Joomla**	**Plone**	**SCMS Flash**	**Share Point Server**
	Advanced Caching	Yes	Yes	Yes	Yes	No	Yes
	Database Replication	No	No	No	Yes	No	Yes
	Load Balancing	Yes	Yes	No	Yes	No	Yes
	Page Caching	Yes	Yes	Yes	Yes	No	Yes
	Static Content Export	Yes	No	No	Free Add-On	No	Yes

Source: CMS Matrix.2007. The Compare Stuff Network. 15 August 2007 <http://www.cmsmatrix.org>

Based on Table 2.4, SCMS Flash provides no performance tuning capabilities while MS SharePoint Server and Plone implement all such capabilities. Lenya, Drupal, and Joomla follow Plone to respectively rank third through fifth. Drupal and Joomla do not support the Database Replication and Static Content Export (SCE) features. Joomla

does not support Load Balancing either. However, based on the data available from EZOSHosting, both content management systems are observed to meet 100% of their requests for moderate and high load bench marking ("Benchmarking").

7. Deployment: For the selected content management systems for this study, deployment is tedious and difficult. The most difficulties are in configuration and getting dependencies resolved. Once all required infrastructures, dependent platforms, and other relevant required facilities are up and running, users could configure their systems. The configuration process is tedious and time consuming. The configuration process becomes more tedious especially in the case of Drupal, Joomla, and SharePoint Server ("Installation," Plone), ("Apache Lenya 1.2 Install," Apache Lenya), ("Install Drupal," Drupal), ("Installation Guide", Joomla), ("Install Office," Microsoft TechNet).

8. Adaptability: Based on the definition of adaptability as introduced in Section 2.3, none of the content management systems in this study are adaptable enough. However, MS SharePoint Server enjoys some degrees of adaptability because it is an enterprise CMS. All other content management systems could be used for web domain only.

9. Stability: None of the content management systems in this study are stable based on the definition of the criterion in Section 2.3.

10. Accessibility: None of the content management systems in this study could satisfy the accessibility criterion as defined in Section 2.3. MS SharePoint CMS is an enterprise and uniform system that could be configured to set users contents accessible across various domains such as document management, record management, and forms

management. MS SharePoint Server could manipulate contents of various formats. Since Joomla, Drupal, Plone, SCMS Flash, and Apache Lenya are not enterprise systems and their vendors are mostly focused on the web domain, the accessibility capabilities for these systems are very poor.

10. Integration: Based on the definition of this criterion in Section 2.3, none of the content management systems in this study except for MS SharePoint Server meet this special criterion. This is due to the fact that Joomla, Drupal, Plone, SCMS Flash, and Apache Lenya are not enterprise systems nor do their vendors offer compatible products for use in any domain other than the web domain. This is because such content management systems are application and domain dependent and do not support interfaces with other content management systems that cover different domains. MS SharePoint Server also has certain limitations: it is limited only to the Microsoft Windows platform. However, major organizations traditionally use Linux and UNIX for their development activities.

It might be argued that it would be a biased judgment to state that since web content management systems do not support other fields of managing content, they lack integration capacity. However, this objection is not valid because the systems which are only focused on a specific domain and do not have any interface with other domains may not be considered complete systems.

2.5. Analysis

This comparative study shows the ensuing weaknesses and the points of strength of the selected content management systems using the criteria stated and described in Section 2.3. The six different content management systems are: Joomla, Plone, MS SharePoint Server, SCMS Flash, Apache Lenya, and Drupal. In Section 2.5, the results of the comparative study that were presented in the previous sections are overviewed, the strengths are mentioned and the problems are highlighted for these content management systems. Then the identified problems are categorized for further elaboration and discussion.

For the criteria mentioned in Section 2.3, four of the content managements systems under this study (Joomla, Plone, MS SharePoint, and Drupal), were fairly applicable for their domain by providing various popular features for polling, picture galleries, e-commerce, blogs, chats, and search engines. Apache Lenya and SCMS Flash lagged behind with the least number of capabilities and built-in applications support.

Content management systems in this study are limited in their extensibility. Joomla, Plone, Lenya, and Drupal are extensible in one direction: users can add alternative or new capabilities. New capabilities cannot be added without support from the development framework or changing the software application. MS SharePoint Server and SCMS Flash are not really extensible. None of the six content management systems are capable of extending the infrastructures such as web server and database servers that they depend on through their frameworks. Therefore, the extensibility is limited for all.

Joomla, Plone, Lenya, and Drupal are free under GPL. MS SharePoint Server for web purposes is very expensive and prohibitive. Enterprise content management systems such as MS SharePoint Server are expensive in general due to their support for many capabilities and functionalities. However, many features of such systems would remain unused due to inadaptability and the lack of interest. Therefore, the return on investment on such systems is generally low.

In terms of security and performance, Plone is the pioneer followed by MS SharePoint Server. Security wise, Joomla and SCMS Flash have poor capabilities, while the rest are equipped with various security techniques. Performance wise, all content management systems in this study, except SCMS Flash, were shown to have capabilities that positively impact the systems' performances.

As describe above, for applicability, security, flexibility, and performance criteria, Joomla, Plone, Lenya, MS SharePoint Server, and Drupal have moderate capabilities, but they are limited in their extensibility features. Moreover, they all do poorly for deployment, adaptability, stability and accessibility. In terms of cost, commercial licenses for content management systems are quite expensive.

Based on this comparative study, the identified problems could be categorized into six groups:

- The existing content management systems are not quite adaptable. This means that current content management systems cannot function in domains other than their native domains.

- The available content management systems are limited in extensibility as there is no support for extending the infrastructure.

- Current content management systems are not flexible to satisfy users' needs in various terms. Moreover, they cannot keep up with changes in emerging technologies. Specifically, content management systems' limitations in extensibility will hinder their flexibilities to support new infrastructures and cope with technological changes.

- The available content management systems have short life-spans as their inflexibility to deal with changes will not let these applications live long and survive the waves of technological changes.

- The only content management systems that could be used across different domains and yet be fully integrated are the enterprise content management systems. The enterprise content management systems; however, suffer from some inflexibilities and will not be effectively used as established above. This would result in shorter returns on customers' investments.

- The existing content management systems cannot meet the needs of larger organizations as they cannot be easily integrated.

The weak points and the points of strengths of the selected content managements systems were highlighted in this study, and a valid solution that could rectify the problems with stability, adaptability, extensibility, accessibility, and deployment would be very valuable.

2.6. Conclusion

The existing content management systems have moderately good records with subjects such as security, applicability, and performance which are related to techniques used in software development. Therefore, most content management systems under this comparative study do relatively well at a technical level. However, these content management system applications seem to suffer from design related problems as the symptoms such as issues with adaptability, stability, and extensibility are not technical issues. Such symptoms are due to insufficiencies and problems with designs or design models.

A content management software system that could be technically equivalent or better and that could address the design related issues such as stability, adaptability, and extensibility would be of great value.

The software stability model, as introduced in Chapter One, offers lucrative and result-oriented solutions to the stated problems. In fact, SSM guarantees full software stability and adaptability by separating the application knowledge layer from the rest of the application code; thus, making a software system adaptable enough to be used in any domain or environment. The resulting software system would be stable. This is because changes in software specifications shall not require changes in the content management engine which is designed using the software stability model.

Furthermore, the stable solution offered by the software stability model could simply use any technique at the application level as an IO and set comparable

benchmarks in applicability, security, flexibility, and performance when compared with the existing content management systems.

Chapter Three: UCME's Basic Building Blocks

3.1. Introduction

This chapter presents the most basic building blocks of a unified content management engine based on the software stability model.

In Chapter One, the Software Stability Model (SSM) approach in extracting application knowledge from the entire application was introduced and discussed. The first procedural step in using the software stability model is identifying and recognizing the ultimate goals of an application. Each goal is called an Enduring Business Theme (EBT) by the software stability model's terminology. An EBT represents a robust and fully stable concept. A stable concept is a notion that is unchanged over time. After identifying the underlying goals for each identified EBT, a pattern which involves the concept and captures the EBT's surrounding capabilities would be constructed. These constructed patterns around an application's goals are called analysis patterns. The software stability model further identifies other design components that are meant to empower the recognized EBTs. These components are termed Business Objects (BO) in SSM. BOs are capabilities that are considered to be internally stable and externally adaptable. After identifying all EBTs and BOs and developing the analysis patterns, for each BO a pattern is developed. The patterns that are created around the BOs are called design patterns.

Next in the design process, the analysis and design patterns are all placed together. Then, each goal in an analysis pattern is mapped to all relevant BOs or

capabilities. This would make it easy to identify all the required BOs for a specific EBT in mind. The connected analysis and design patterns create a whole system which is known as knowledge map. The next step would be constructing the engine.

In Chapter Three, the goals or EBTs and the capabilities or BOs are identified for design of a content management system. Next, analysis and design patterns are developed and a knowledge map is constructed. This is followed by further discussions on some relevant development issues and deployment factors. More details about the design would be discussed in Chapter Four.

Table 3.1 contains a comprehensive list of all goals and capabilities that this research considers being involved in design of a unified content management engine. The list is quite large and extensive as the goals and capabilities that a content management system should support are quite numerous. The goal of a content management system is not only organizing the content. The term content management is rather stretched to include capabilities that are normally thought to be outside the definition of management. As correctly put in by The Gilbane Report, "there are even more things to do to content that might be, and are, considered 'managing' (authoring, acquiring, publishing, dynamic page generation, integrating, assembling, versioning, configuring, linking, delivering, caching, analyzing, sharing, searching, categorizing, transforming, re-using, syndicating, archiving, etc.)" ("What Is Content Management?" 3). Table 3.1 lists many recognized goals for a content management system.

Table 3.1

The Goals and Capabilities Identified for CMS Engine

Term	Type	Description
Managing	EBT	Includes Monitoring, Directing, Controlling, Planning, Staffing, which are captured within Management
Monitoring	EBT	To observe or check on something or someone
Planning	EBT	To identify, arrange, and schedule activities
Controlling	EBT	To operate, restrain, or exercise power over something
Staffing	EBT	To bring and organize actors or parties in for a task
Organizing	EBT	To arrange components to create a specific structure
Optimizing	EBT	To enhance something for a specific functionality
Assigning	EBT	Attaching or relating one entity to another
Changing	EBT	Modifying
Directing	EBT	To organize and oversee a flow of tasks or a structure
Visualizing	EBT	To bring things into pictures or any visual form
Customizing	EBT	To alter or change in order to fit into an expectation
Authorizing	EBT	To grant permissions to do some tasks
Ownership	EBT	To own something
Authenticating	EBT	To confirm credentials or the truth of something
Personalization	EBT	To customize for an specific account
Linking	EBT	To relate, tie, or attach
Ordering	EBT	Organizing entities by specific rules
Classifying	EBT	To categorize things in groups based on a rule or commonality among the things being classified
Measurement	EBT	Determining the magnitude or size of something
Comparing	EBT	To examine entities for similarities or differences
Sharing	EBT	To use or own entities in common with others

Table 3.1 continued...

Validating	EBT	To verify the correctness of something stated
Extending	EBT	Stretching to increase the size or coverage
Accessing	EBT	To reach any entity
Delivering	EBT	To bring a task or an entity to a destination or conclusion
Searching	EBT	To examine and look into, through, or over something to find an entity or a thing of interest
Dynamism	EBT	Automated change based on a rule
Authoring	EBT	Composition and creation of a written thing such as a document, program , musical note, etc.
Versioning	EBT	To mark something with a label for managing, organizing, referencing, archiving, etc.
Configuring	EBT	Adjusting and tuning a system for a setting
Archiving	EBT	To store for future references
Integrating	EBT	To merge components and parts into a larger whole or a system
Editing	EBT	To modify a written document, music, video or content
Analyzing	EBT	To examine an entity or something in details
Profiling	EBT	Classifying and analyzing an entity or a person, based on the identity of the person or the characteristics of the entity
Publishing	EBT	Preparing and delivering
Assembling	EBT	Mounting and putting entities together
Arranging	EBT	Placing things based on an order
AnyContent	BO	Any form of data or information like text, music, video, photo, etc.
AnyMetric	BO	A quantitative system for measurement
AnyMedia	BO	Represents a media for communication or interaction such as wired line, wireless network, radio signals, etc.
AnyMechanism	BO	Represents a mechanism for performing a task
AnySetting	BO	Represents an state of configuration

Table 3.1 continued...

AnyCapability	BO	Represents the capabilities of an entity or the goals that an entity could achieve, such as capability of lifting certain weights, refreshing at specific rate, etc.
AnyArrangement	BO	Represents the placement of items, or how entities or components are placed next to each other
AnyProject	BO	Represents a project
AnyStructure	BO	Represents the relations among components of a system, like structure of a page (also known as AnyLayout)
AnyCategory	BO	This class represents any category or classification
AnyCriterion	BO	It represents any rule or standard upon which decisions could be made. This also represents specifications.
AnyOrder	BO	Represents a way or rule that items or entities are arranged
AnyVersion	BO	Represents a version
AnyStandard	BO	Represents an standard or rule for doing things
AnyEntity	BO	Represents any entity in design
AnyContainer	BO	Represents a storage for any content
AnyPresentation	BO	Represents a visualized content
AnyIdentity	BO	Represents identity of something especially for authenticating and authorizing
AnyReference	BO	Anything that could be referenced or used back
AnyParty	BO	Any user or entity with contractual capacities
AnyLog	BO	Represents logs of contents in design
AnyService	BO	Represent a service that could be provided
AnyResponsibility	BO	Represents a set of duties and services that one is obliged to provide
AnyAttribute	BO	Represents attributes or properties of something
AnyType	BO	Represents a kind or brand of something
AnyState	BO	Represents a state of an entity or thing
AnyTool	BO	Represents any tool or software
AnyView	BO	Represents a view or display of an entity

3.2. Goals Assessment

As stated in the previous paragraphs and chapters, the first real step in software design through the software stability model is to identify the ultimate goals of an application. The goals are also termed EBTs. They are enduring business themes and fully stable concepts, which do not change over time. Table 3.1 above lists many recognized goals for a content management system. Evidently, there are many goals that need to be captured by a content management system as the notion of managing content is really stretched to include other concepts such as publishing, editing, creating, sharing, authorizing, versioning, archiving, or analyzing. When the goals are finally identified, they would be used to develop analysis patterns around them and later empower them with capabilities that are needed to drive them. This is carried out through extensive empirical studies, searching, and verifications to capture the core capabilities around the EBTs. This extensive work consumes days and hours in studies and discussions. In this section and part of this thesis report, two of the major goals and their associated analysis patterns are presented. The significance of these two EBTs is that they would resolve the inflexibility of current content management systems as described in Chapter One and Chapter Two. These EBTs are Dynamism and Personalization. The following sections describe the names of these analysis patterns, the problems and challenges in using the patterns, the contexts they could be used in, and finally the solution which is the class diagram of the analysis patterns.

3.2.1. Dynamism Analysis Pattern

The Dynamism Analysis pattern basically explains the concept of dynamism. The pattern could be used for making websites, presentations, texts, or any dynamically changing content. It could bring dynamism in many applications in which employing dynamism is deemed particularly necessary.

- **Name:** The name of this pattern is Dynamism.
- **Context:** The Dynamism analysis pattern could be used in various and different contexts of interest. In this research, the main interest is in making representation of contents truly dynamic. At the minimum, an efficient mechanism is required for making an entity dynamic through changing its state to a new state.
- **Problem:** The Dynamism analysis pattern could be used in various scenarios. However, covering all possible scenarios in which dynamism would play a role requires capturing the circumstances seen in all such scenarios, which is a real challenge. Moreover, a design that covers all such scenarios would also require complex and tedious works.
- **Solution and participants:** Figure 3.1 depicts the class diagram for Dynamism analysis pattern using the Unified Modeling Language (UML).

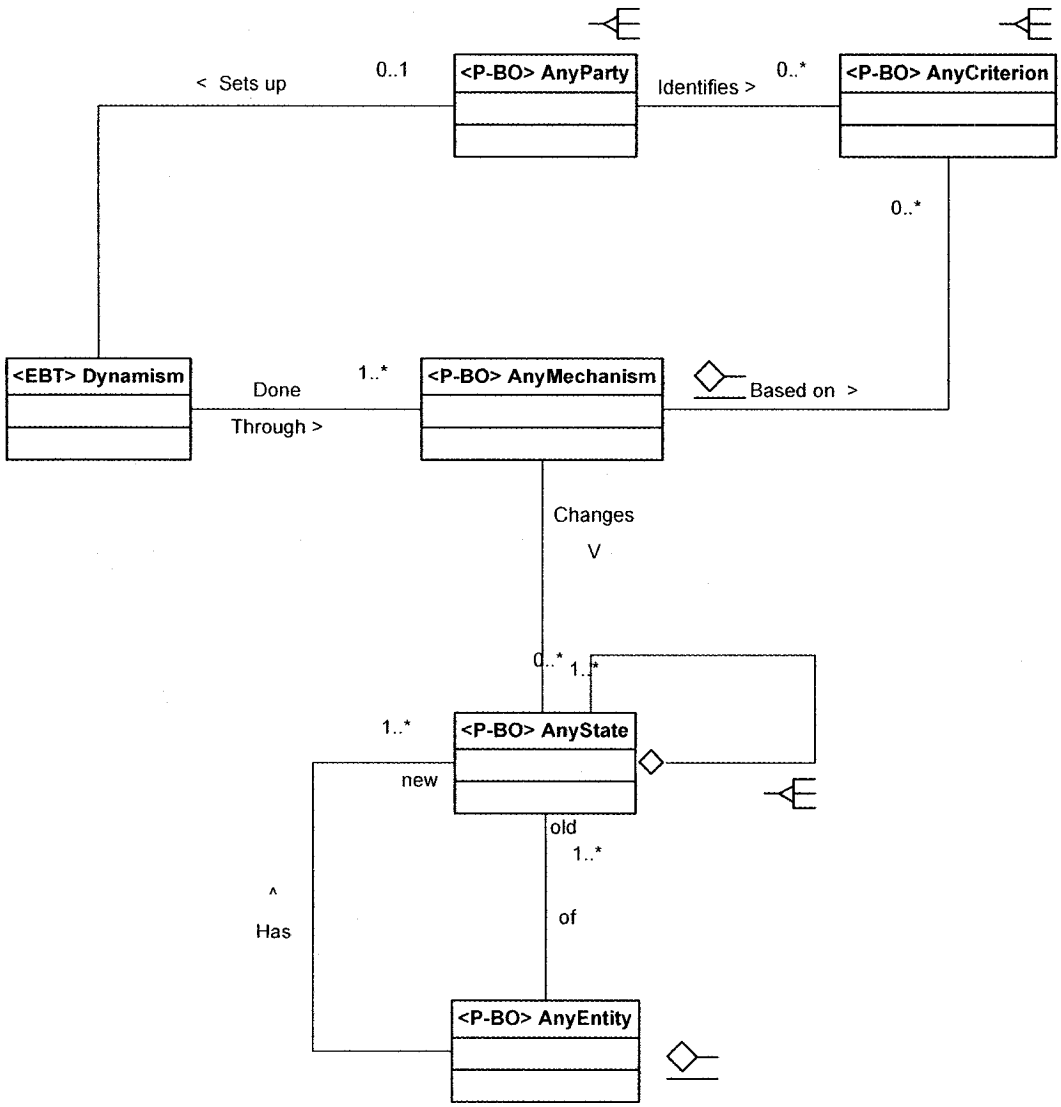

Figure 3.1. Class diagram for Dynamism analysis pattern.

3.2.2. Personalization Analysis Pattern

Personalization is referred to as configuring and customizing components of a system, such as a web site, a room, or an object for someone or something.

• **Name:** The name of this analysis pattern is Personalization.

• **Context:** The Personalization analysis pattern could be employed in and later applied to various contexts. In this work, personalizing content is of immense interest. At the minimum, the contexts for which personalization could be applied must consider and include a role for any party that would want to personalize an entity. Moreover, at least one mechanism for personalization must be present along with the entity to be personalized.

• **Problem:** The major challenges in defining the Personalization analysis pattern rest on capturing the various circumstances in many scenarios of personalization.

• **Solution and participants:** Figure 3.2 depicts the class diagram of the Personalization analysis pattern.

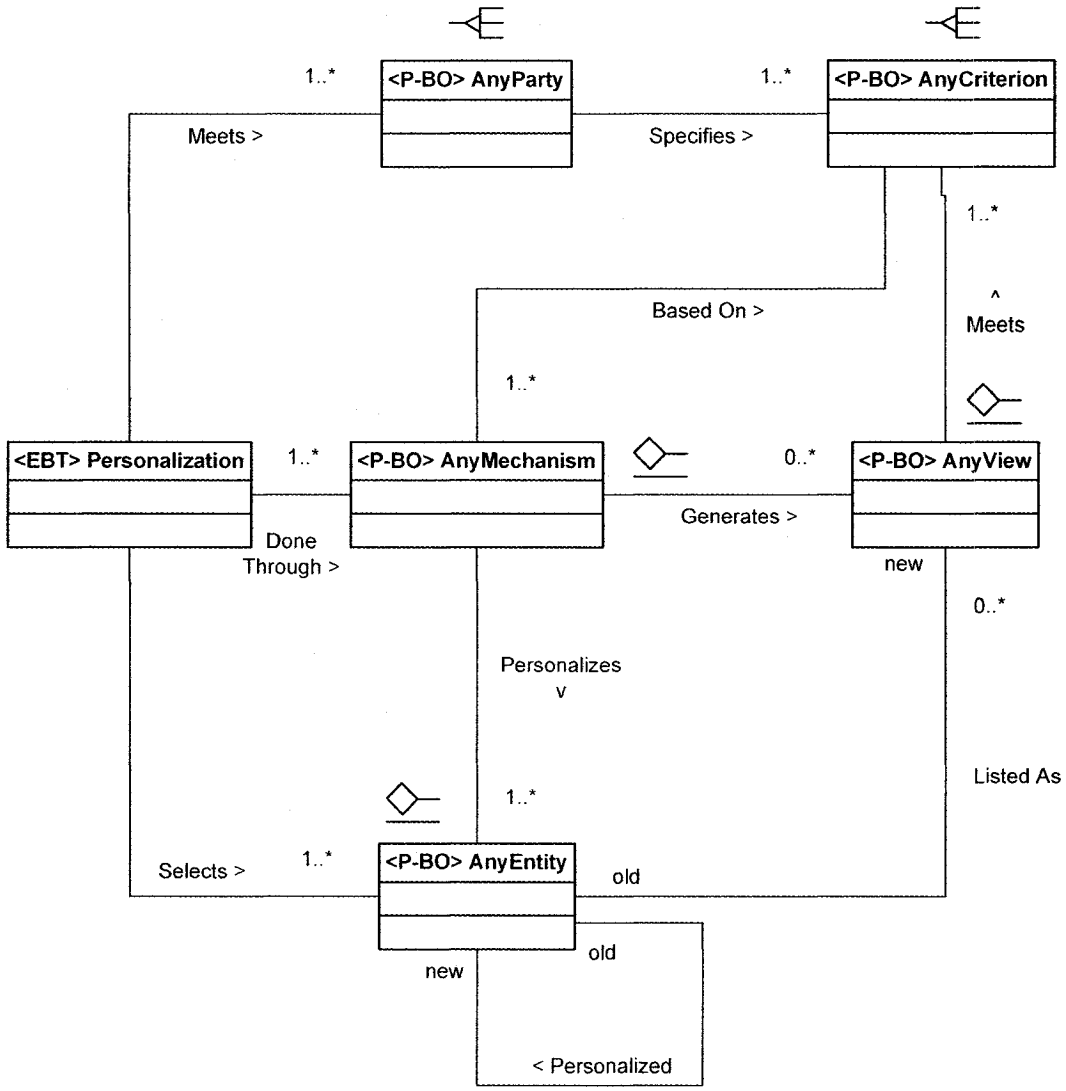

Figure 3.2. Class diagram for Personalization analysis pattern.

3.3. Capabilities Assessment

As explained in Chapter One, capabilities in the software stability model are classes that represent concepts or things that empower EBTs in design. Capabilities are termed BOs in SSM and are considered to be internally stable and externally adaptable.

An Industrial Object (IO) represents an application's component. BOs in design could be connected to EBTs, other BOs, and some IOs. In this thesis report, BOs that are identified for content management engine are listed in Table 3.1. After identifying the BOs, the second layer of EBTs and BOs that could empower the identified BOs would be considered and a design pattern would be developed for each identified BO. In this section, three design patterns are introduced: AnyContent, AnyEntity, and AnyPresentation.

3.3.1. AnyContent Design Pattern

AnyContent design pattern captures the knowledge around the idea of content. This design pattern is of great importance for a content management engine's design. The content that this pattern represents could be in forms of a document, photo, music, or video. The content could be in some structured form such as TCP/IP Meta data.

- **Name:** The name of this design pattern is AnyContent.
- **Context:** AnyContent design pattern could be used in any context in which the notion of content has a definite role. It could be editing a document, playing a song, or composing a movie. At the minimum, any content should capture some knowledge and reside on a media. Any content also, must have a log in some forms such as text, photo, or sound.
- **Problem:** The AnyContent design pattern could be applied to different scenarios. However, covering all possible scenarios in which AnyContent would play a role requires capturing various settings in all such scenarios. This is a difficult task and a great

challenge. Moreover, working with various forms and types of content is another challenge.

• **Solution and participants:** Figure 3.3 depicts the class diagram of the AnyContent design pattern.

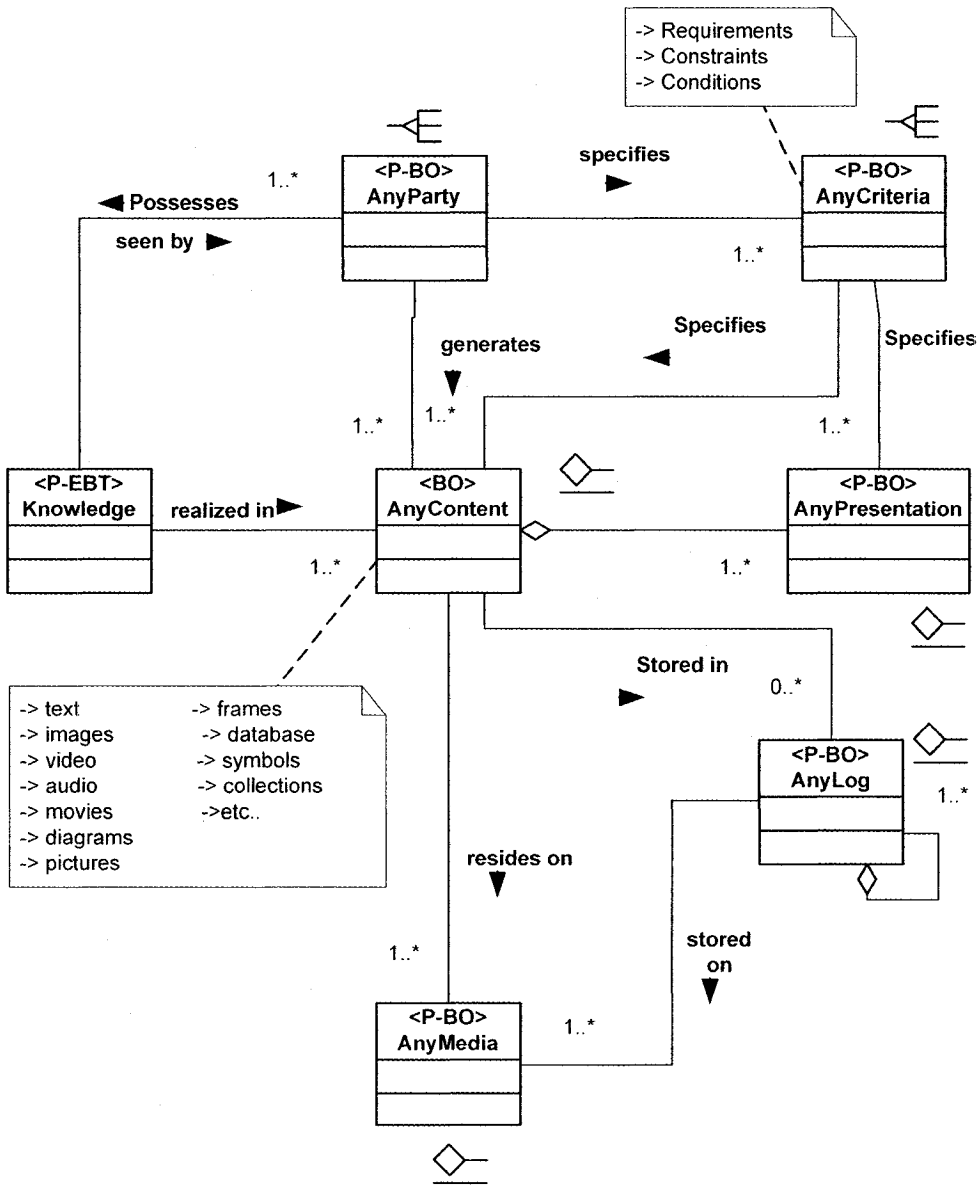

Figure 3.3. Class diagram for AnyContent design pattern.

3.3.2. AnyEntity Design Pattern

AnyEntity design pattern represents any conceptual or physical entity.

- **Name:** AnyEntity
- **Context:** AnyEntity design pattern could be used in different contexts of interest where an entity is to be represented.
- **Problem:** AnyEntity design pattern could be applied to many scenarios or situations. However, covering all possible scenarios in which AnyEntity would have a role requires looking closely into all such scenarios, which is a big task. Moreover, a design that covers all such scenarios would also involve complex and tedious works.
- **Solution and participants:** Figure 3.4 depicts the class diagram of the AnyEntity design pattern.

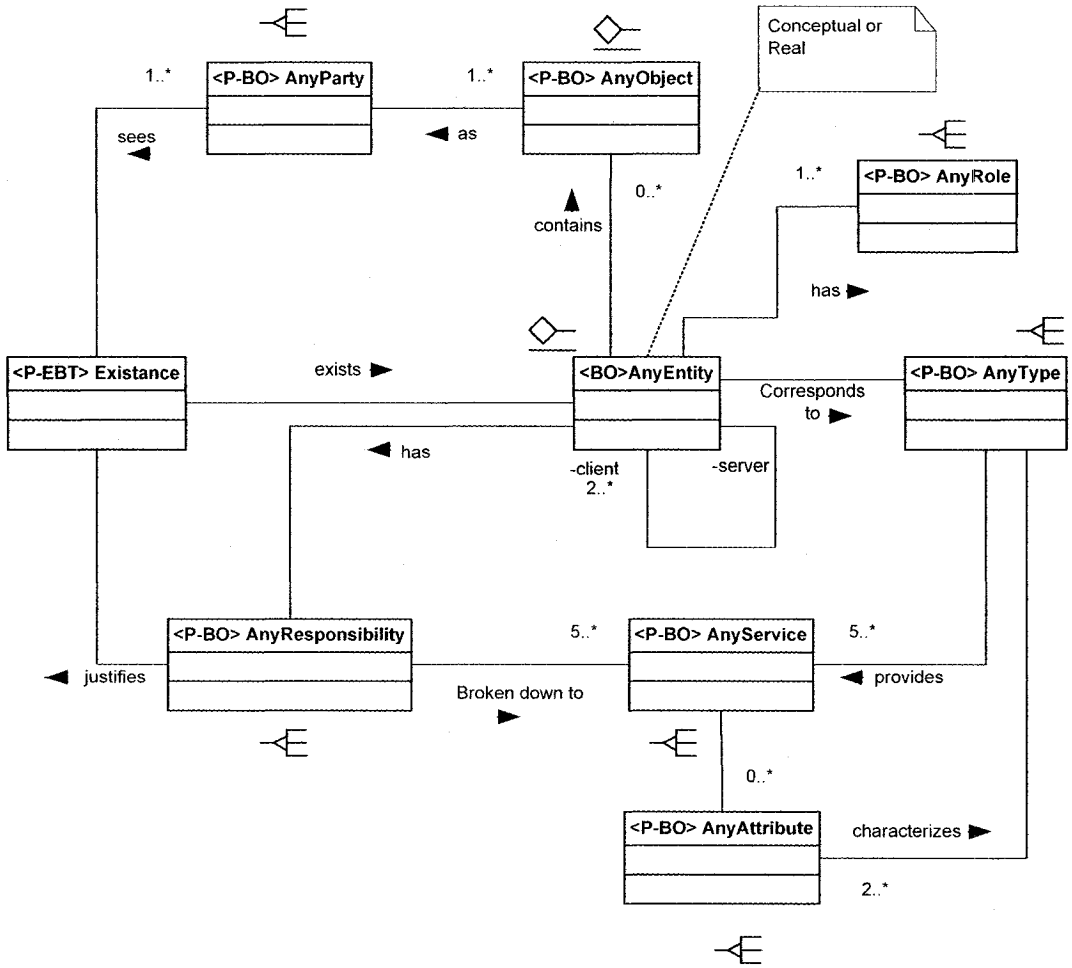

Figure 3.4. Class diagram for AnyEntity design pattern.

3.3.3. AnyPresentation Design Pattern

AnyPresentation design pattern represents a presentation or exhibition of content. It could be presentation of a movie, a photo, or a document.

• **Name:** The name of this design pattern is AnyPresentation.

• **Context:** The AnyPresentation design pattern could be used or employed in different contexts of interest where the main intent is to present, exhibit, or show some content like photos.

• **Problem:** The AnyPresentation design pattern could be applied to many scenarios. However, covering all possible scenarios of visualizing some content requires looking closely into all such scenarios. This is certainly a big challenge. Moreover, a design that covers all such scenarios would be complex and difficult to implement.

• **Solution and participants:** Figure 3.5 depicts the class diagram of the AnyPresentation design pattern.

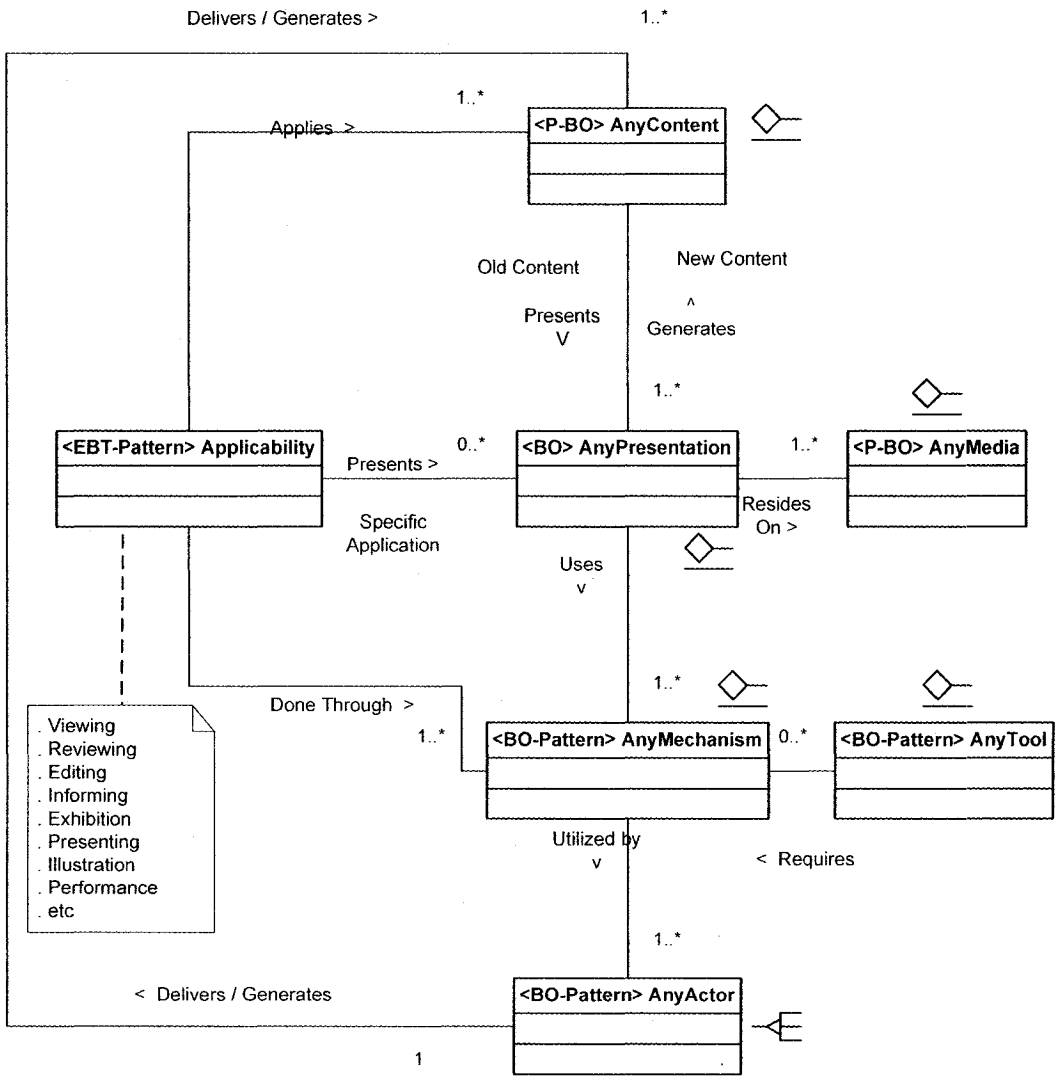

Figure 3.5. Class diagram for AnyPresentation design pattern.

3.4. Knowledge Map: Introduction

As explained in the prior chapters and sections, one of the main characteristics of the software stability model, which injects stability in design, is the idea of separation of knowledge from the rest of the application logics. The software stability model deals with application knowledge by identifying, recognizing, capturing, and relating the various

components of knowledge. In Section 3.4, the process of relating the bits and pieces of some captured knowledge through the use of knowledge maps would be presented. The essential goal of mapping the captured knowledge in the software stability model is to see "how to organize them, and how to relate them to formulate an accurate solution in contexts [8]." In other words, the knowledge map helps us to organize the captured knowledge and relate its various components. Establishing such relations would especially be useful as the software stability model always advocates goal-oriented design approaches. In this case, having certain goals in mind for building an application, a knowledge map could help to easily identify the required capabilities for the goals in mind. This approach would also assist us to create meaningful architectures that would contain only the required components.

As mentioned in the previous paragraphs, the software stability model identifies and captures application knowledge in the form of analysis patterns and design patterns. It was further established that each pattern includes two layers of application independent knowledge components termed EBTs and BOs. In addition, it includes an extra layer that is application-specific and is termed IOs. Figure 3.6 represents this concept in a picture.

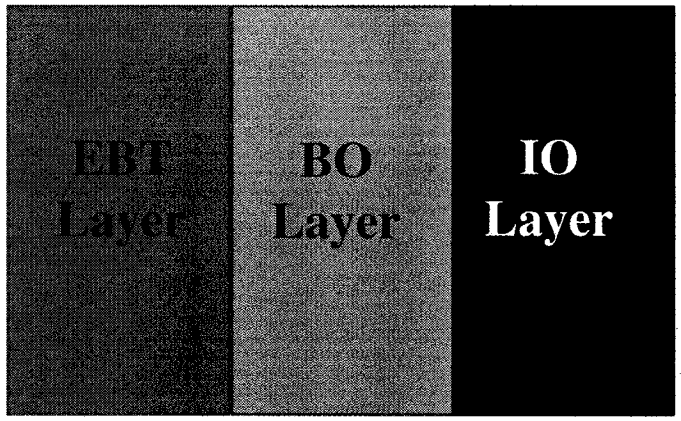

Figure 3.6. Representation of three layers of knowledge captured in a stable pattern.

Each pattern is consisted of several EBT and BOs as shown in Figure 3.7 below.

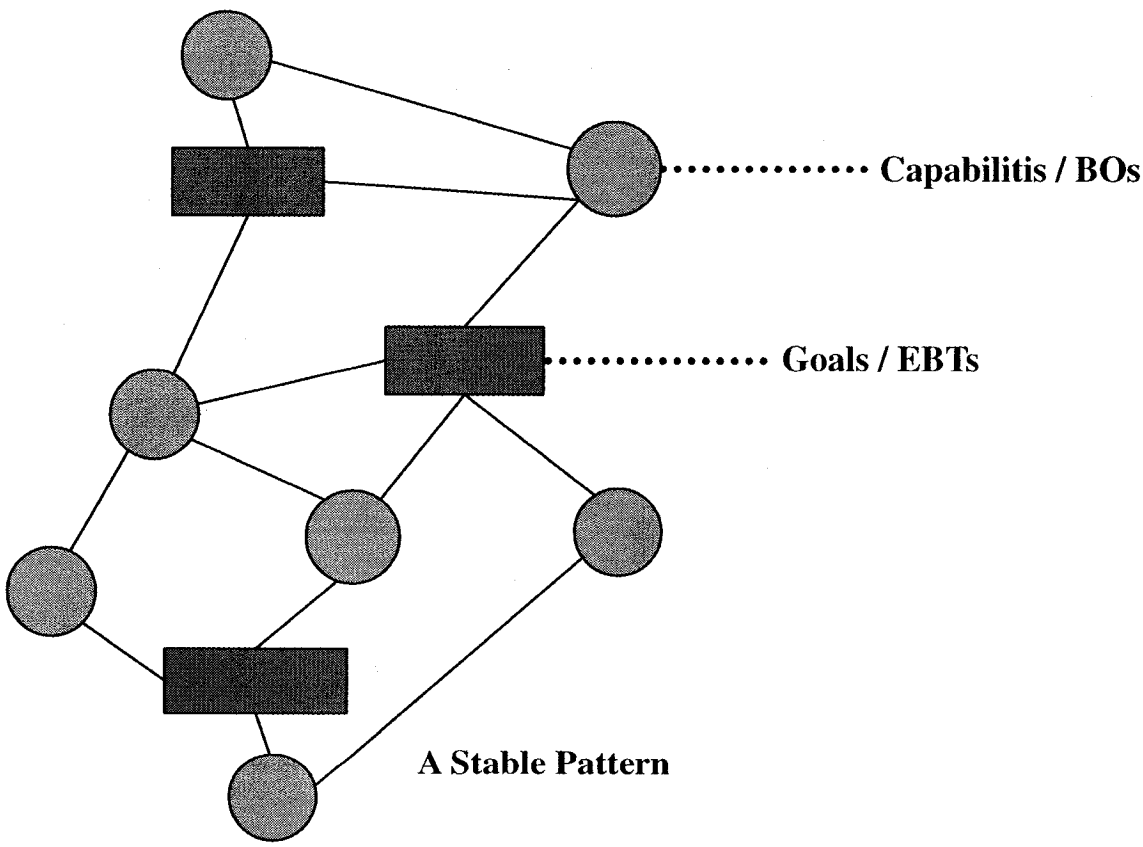

Figure 3.7. Representation of a stable pattern and its components.

In Figure 3.7, the EBTs or goals are depicted by rectangles, and capabilities or the BOs are depicted by circles. The whole figure represents a stable analysis or design pattern.

Captured patterns could be used or targeted for various domains. A knowledge map defines the relationships and associations among patterns that are in the same or other domains. Figure 3.8 depicts this idea.

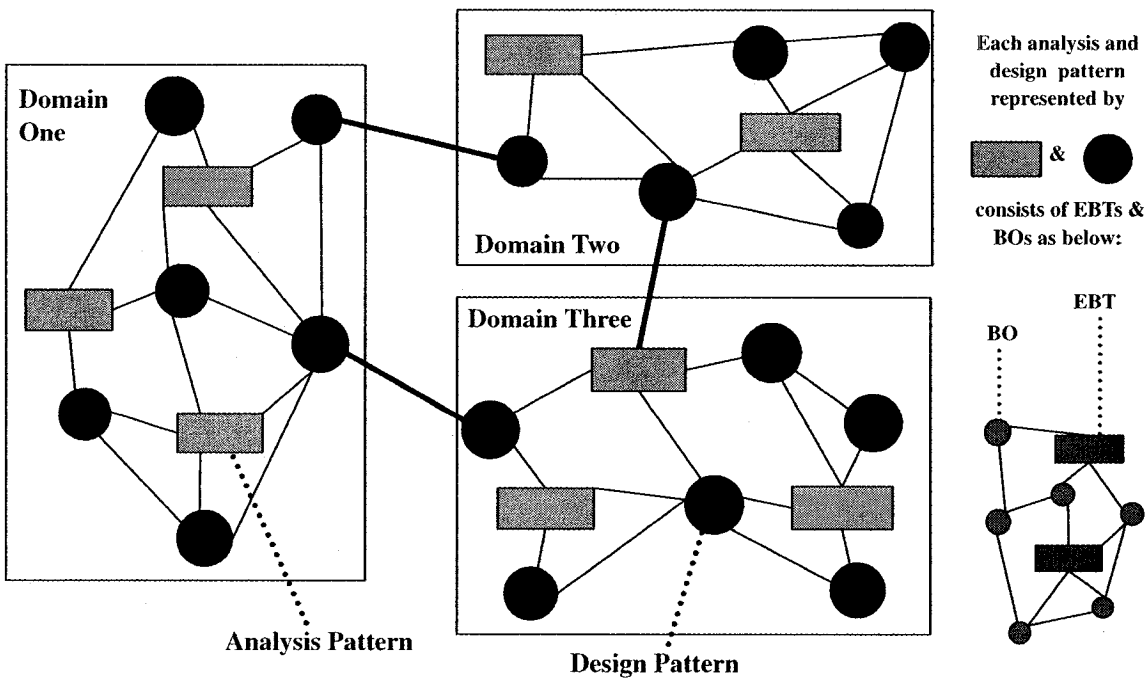

Figure 3.8. Depiction of interconnected patterns in three different domains.

The khaki boxes in the Figure 3.8 represent some analysis patterns, and black circles represent some design patterns. All patterns are captured inside a rectangle that represents an application domain. All patterns are connected inside each domain and also

across the domains. Each analysis or design pattern, in turn, is consisted of some EBTs and BOs as depicted on the right side of the Figure 3.8. Indeed, by mapping certain goals to a set of capabilities one could effectively create architectures or application engines that are meant to satisfy those goals in mind. Choosing certain goals and connecting the analysis patterns that contain the goals through different BOs would yield different application engines. Therefore, various different engines could be constructed by choosing to include different components in a knowledge map. Figures 3.9 and 3.10 represent different architectures.

In Figure 3.9, the red components represent the chosen goals and capabilities that when related, they would construct a software architecture. Such patterns also would satisfy the goals embedded in analysis patterns represented by red rectangles.

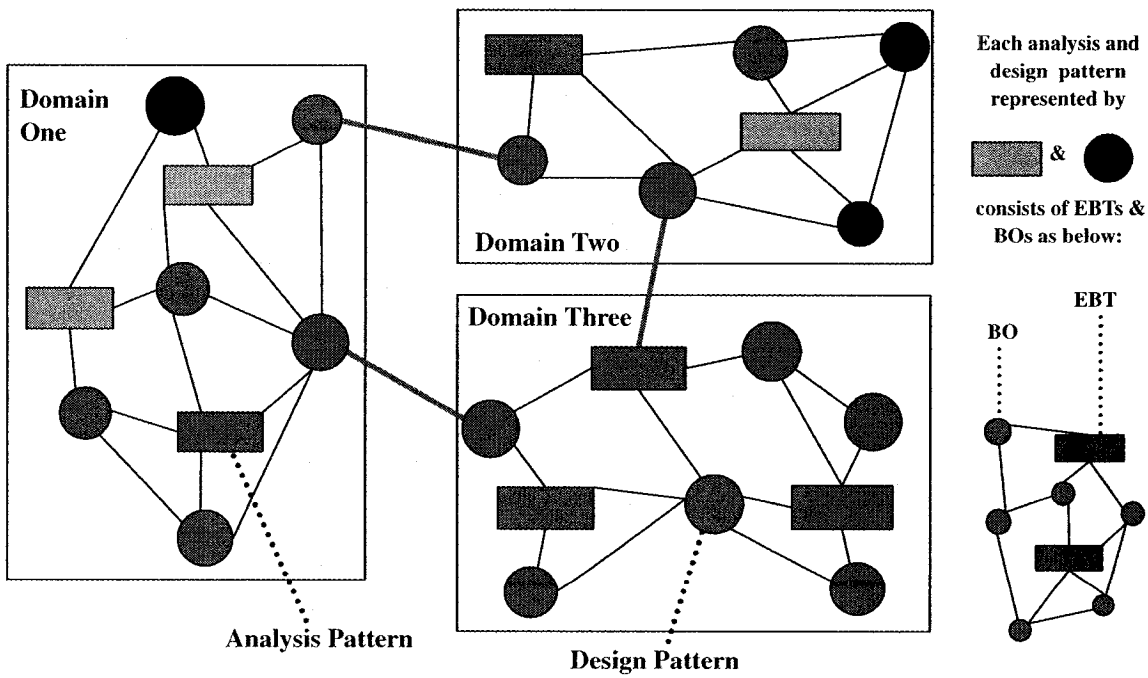

Figure 3.9. Depiction of interconnected patterns that construct pattern architecture.

Figure 3.10 represents another architecture that uses the same domains and patterns as presented in Figure 3.9.

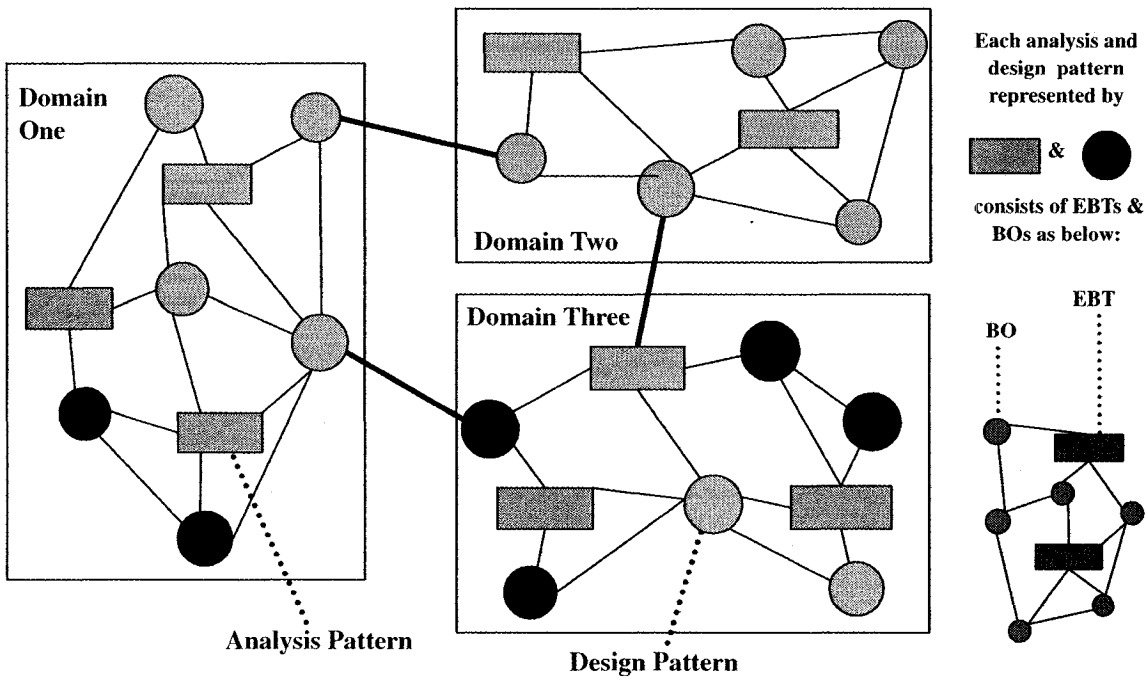

Figure 3.10. Depiction of interconnected patterns in three different domains.

By comparing Figure 3.9 and Figure 3.10, it could be observed that various architectures could be created by choosing different sets of paths to connect the patterns inside and across different domains. The relationships among patterns are captured by the knowledge map.

Other facts about knowledge maps are presented next:

• Fayad and Sanchez indicate that knowledge maps are defined for various aspects of stable software development and satisfy various concerns: "Analysis concerns called

Goals, Design concerns called Capabilities, Development concerns called Development scenarios, and Deployment concerns called by the same name" ("Knowledge Maps").

• Knowledge maps draw the relationships among several patterns and their components. Knowledge maps do not include any pattern or any of its elements.

• Various architectures could be drawn by taking different routes that connect certain EBTs and BOs.

• Fayad and Sanchez note that knowledge maps allow "rational traceability" of design and implementation components along the connecting routs to the goals ("Knowledge Maps").

• The intersecting points on a knowledge map are pattern elements that are shared among two or more knowledge maps. Fayad and Sanchez note that identifying such intersections would aid designers and developers with more intelligence about the knowledge captured inside an application ("Knowledge Maps").

3.5. Development Factors

Once the analysis and design patterns are prepared for designing a content management engine, a knowledge map for all goals of the content management engine needs to be created. The knowledge map would link the system's goals to some capabilities. For example, to create a content management system engine that is meant for publishing and making some contents dynamic, only the Publishing and Dynamism EBTs would be selected. Furthermore, by using SSM knowledge map all the required capabilities would be identified and a stable architecture would be extracted by

connecting the required capabilities and goals through some well defined paths. After identifying the required EBTs, BOs, and their relations, the created design could be implemented by code. There are several major points about development activities in an implementation phase that will be discussed in Section 3.5.

During an implementation phase, the EBTs that represent the conceptual design goals would be implemented as interfaces which could be more completed and empowered by capabilities or BOs. The BOs must implement the EBT interfaces that they are associated with fully or partially. IOs, however, will contain the application logic and would fully implement the application-specific components of a design. Switching between IO components that could connect to a specific BO would result in changes in the software application's behavior. That is exactly how a software application could be modified without changing the engine.

Many components that could allow a content management system to communicate with some certain facilities are considered IOs. An IO is not part of a stable engine and needs to be prepared and used by users. For convenience, however, certain IOs that are commonly used IOs could be supplied to users, so they would not need to create such IOs. Then, user could simply access the IO from a library of all such modules and simply plug them in for use.

3.6. Deployment Factors

When deploying a constructed engine for an application, naturally, some application-specific modules such as IOs should get connected to the capabilities or the

BOs of the engine. This is performed through the Hook facility. Hook facility is a piece of software that allows advanced and novice users to connect their IOs to some engine BOs through a GUI interface. The Hook facility will generate the classes that extend the BOs and use the supplied IOs. For instance, AnyEntity as a BO could be hooked to an IO of interest such as a web page. The Hook facility is developed by Fayad's group as part of a drive to introduce the concept of stable software design more closely to the public.

3.7. Conclusion

In this chapter, the basic building blocks of the content management engine were reviewed in detail. Building upon the introductory paragraphs about the software stability model presented in Chapter One, this chapter presented the roles of design goals and capabilities and the ways they are used for creating an engine. Furthermore, many goals and capabilities were identified for a unified content management engine, and several design patterns and analysis patterns were also presented. The Dynamism and Personalization analysis patterns that were presented in this chapter were of critical and immense importance as they alleviate the current pain of inflexibility in the existing content management systems. The issue of inflexibility was identified in Chapter One and Chapter Two where various content management systems were reviewed in a comparative study. Moreover, AnyPresentation, AnyEntity, and AnyContent design patterns were presented and discussed. These design patterns are the essential elements of a content management engine. Then, the concept of a knowledge map was introduced and presented with various figures. Ultimately, some development and deployment

factors were identified and explored in this chapter to conclude presenting the basic building blocks of a unified content management engine.

Chapter Four: The Unified Content Management Engine

4.1. Introduction

After introducing the basic construction blocks of a Unified Content Management Engine (UCME) in Chapter Three, the said engine is further reviewed, elaborated and explained in Chapter Four.

It is no surprise that the basic and advanced concepts of the Software Stability Model (SSM) employed in construction of a unified content management engine and its building blocks greatly influence and impact the characteristics and potential of UCME software applications.

As introduced in Chapter One and later reviewed in Chapter Three, the software stability model introduces a unique development model for software applications such that the end result will preserve its integrity and stability over time and change. It was further iterated that the software stability model also introduces the concepts of Enduring Business Theme (EBT), Business Object (BO), and Industrial Object (IO) in a move to separate application knowledge from the application's business logic. By capturing the available knowledge in domains of interest in an abstract way, the software stability model introduces fair amount of stability to the final design. The core knowledge remains unchanged across various domains of applicability. However, the application or domain-specific modules may vary across domains.

As a short review, EBTs define the goals for which an application is being built for. These goals are fully stable and non-changeable. BOs are those capabilities that

empower the EBTs to do what they are meant to do. In the course of applying the principles of the software stability model, as described in Chapter Three, all analysis and design patterns associated with EBTs and BOs are constructed and tied together to compose an engine to capture the available application knowledge. The engine could be deployed, employed, and used in various domains with full stability. IOs are unstable, changeable, application-specific modules that capture application logic.

After the short review and introduction stated above, the rest of this chapter reviews the unified content management engine which was introduced in Chapter Three. Then, the architecture of the engine is overviewed and its other architectural elements are discussed. Next, an application of the engine is set forth and discussed. Next, the impacts of constructing a unified content management system over various domains are discussed. This chapter is then concluded by a short overview.

4.2. UCME Overview

The basic building blocks of the unified content management engine were introduced in Chapter Three. The EBTs, BOs of engine were presented, and two major EBTs and their three related BOs were selected for more deliberations. Then, two analysis patterns were introduced for Personalization and Dynamism EBTs along with three design patterns for AnyContent, AnyPresentation, and AnyEntity. Moreover, the concept of knowledge map was first introduced and then elaborated in details along with various illustrations. Finally, the notion of software architecture as a fundamental concept in the software stability model was presented and described; however, Chapter

Three stopped just short of presenting the UCME architecture specifically. Discussing the architecture of UCME specifically and classifying stable architectures in general are the main themes of this chapter.

4.2.1. UCME Architecture

Chapter Three presented how various patterns could be connected together and mapped via knowledge maps to form some unique architectures. In Section 4.2.1, the approach presented in Chapter Three is used to present the UCME architecture.

The unified content management engine consists of many goals and capabilities due to the nature of the system which is meant to satisfy a myriad of operations such as publishing, presenting, editing, comparing, versioning, adding, deleting, organizing, configuring, and personalizing. In this thesis report, two major goals that could address some major pitfalls of the existing content management systems are used along with their associated capabilities to construct the UCME engine. The goals and capabilities used in the UCME of this thesis are: Personalization, Dynamism, AnyEntity, AnyContent, and AnyPresentation. These concepts are well elaborated and discussed in Chapter Three. Figure 4.1 represents these five analysis and design patterns forming a possible architecture for the content management engine by the methods already introduced in Chapter Three. This architecture is the UCME's unique architecture. This shall be noted that in the Figure 4.1 each rectangle represents a pattern, and each pattern includes several classes. From a pure software implementation perspective, a pattern is essentially a collection of closely related classes.

As illustrated in Figure 4.1, both Dynamism and Personalization analysis patterns interact directly with AnyEntity and AnyContent design patterns, and AnyPresentation interacts with the design pattern of AnyContent. Therefore, both Dynamism and Personalization analysis patterns indirectly act on AnyPresentation design pattern. The main rationale behind such an indirect connection to AnyPresentation design pattern is that Personalization, Dynamism, and AnyPresentation are designed domain independent as they should be able to take part in domains other than the content management domain. Therefore, when a domain of interest is broadened, some components that could exist in a more restricted domain might need to be removed from the design. In other words, patterns become more abstract, and more restrictions in design would be required when designs are meant to target more domains. After all, the concept of software stability advocates for application-agnostic design.

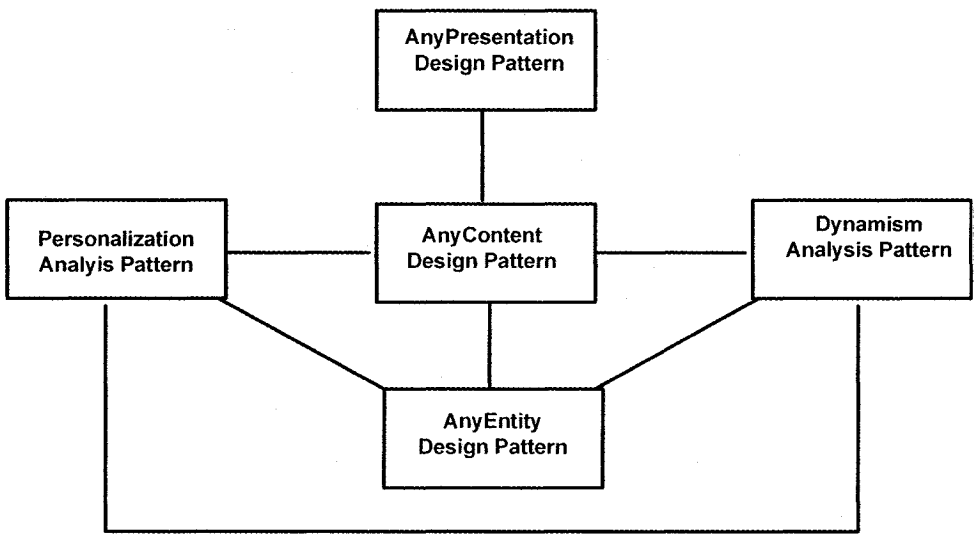

Figure 4.1. A partial UCME architecture.

Figure 4.1 represents UCME architecture in an abstract pattern level. A more detailed view of the figure would mosaic and place the patterns together at a class level. Figure 4.2 illustrates a more detailed view for a section of UCME architecture. This section of architecture is the Personalization analysis pattern at a cross point with the AnyEntity design pattern. In Figure 4.2, the pattern border for Personalization is marked with a blue line, and the limit of the AnyEntity design pattern is marked by a red line. One of the shared nodes between the two patterns is the AnyEntity class, and that is one of the points where the two patterns are connected. Of course, AnyParty is another shared node, but Figure 4.2 is meant to illustrate the concept in a simplified diagram to avoid any confusion. For that, only one shared node is highlighted and marked.

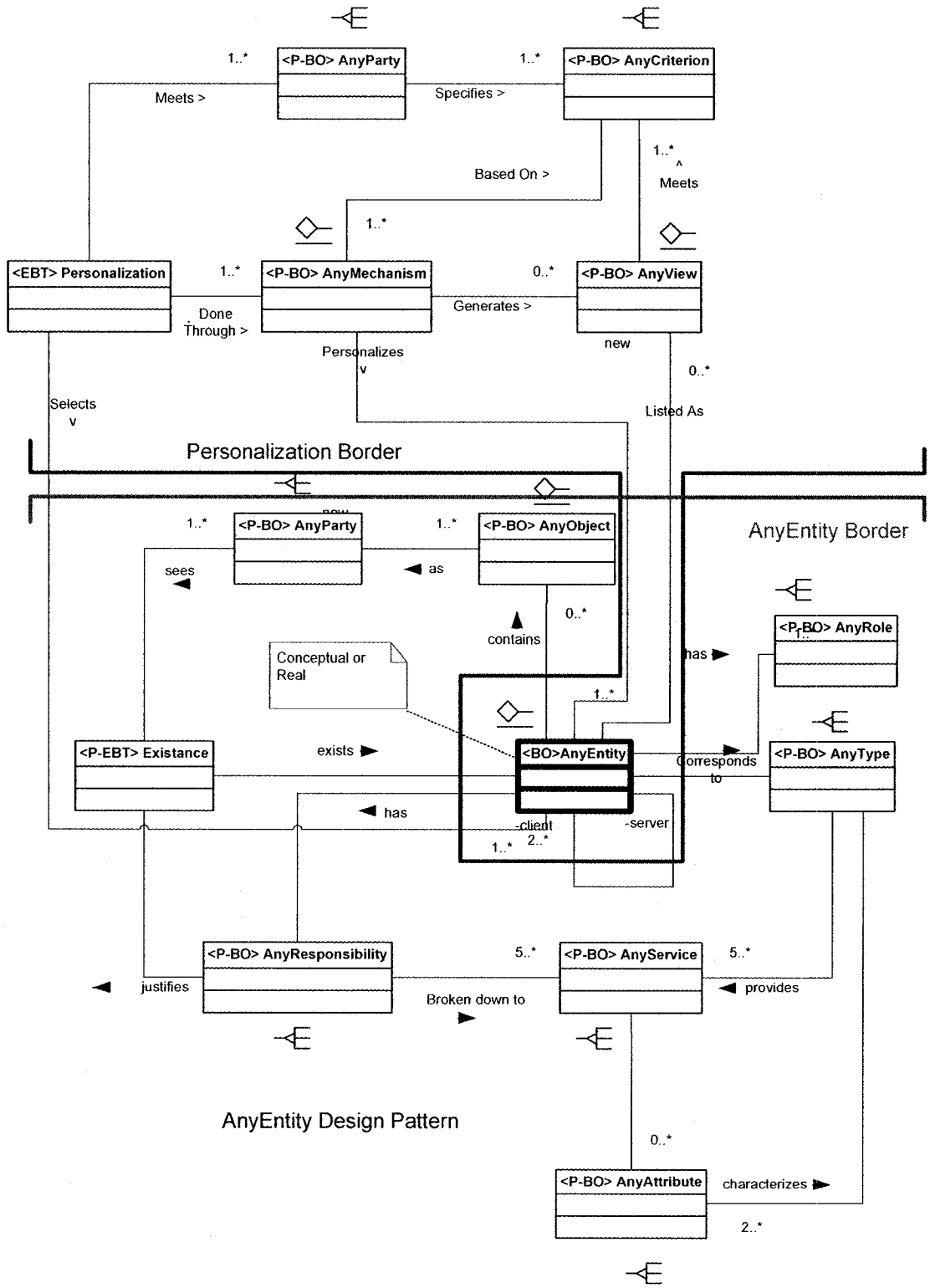

Figure 4.2. Depiction of AnyEntity and Personalization patterns.

Now that the concepts and principles of developing stable architectures by interconnecting analysis and design patterns is presented, it is easier to look at more patterns involved in the UCME design.

As stated in the prior paragraphs, the complete UCME engine is large in size due to the fact that more goals and capabilities are involved in the process of managing contents. Also, it was mentioned that the architecture in Figure 4.1 is a UCME that supports only personalization and dynamism as goals. Naturally, this architecture could be larger in nature if the EBTs and BOs introduced in Chapter Three were used. Figure 4.3 depicts a hypothetic architecture of content management engine with the introduction of two more goals shown in blue color: Monitoring and Organizing. Naturally, the extended architecture is larger than the illustrated architecture in Figure 4.1.

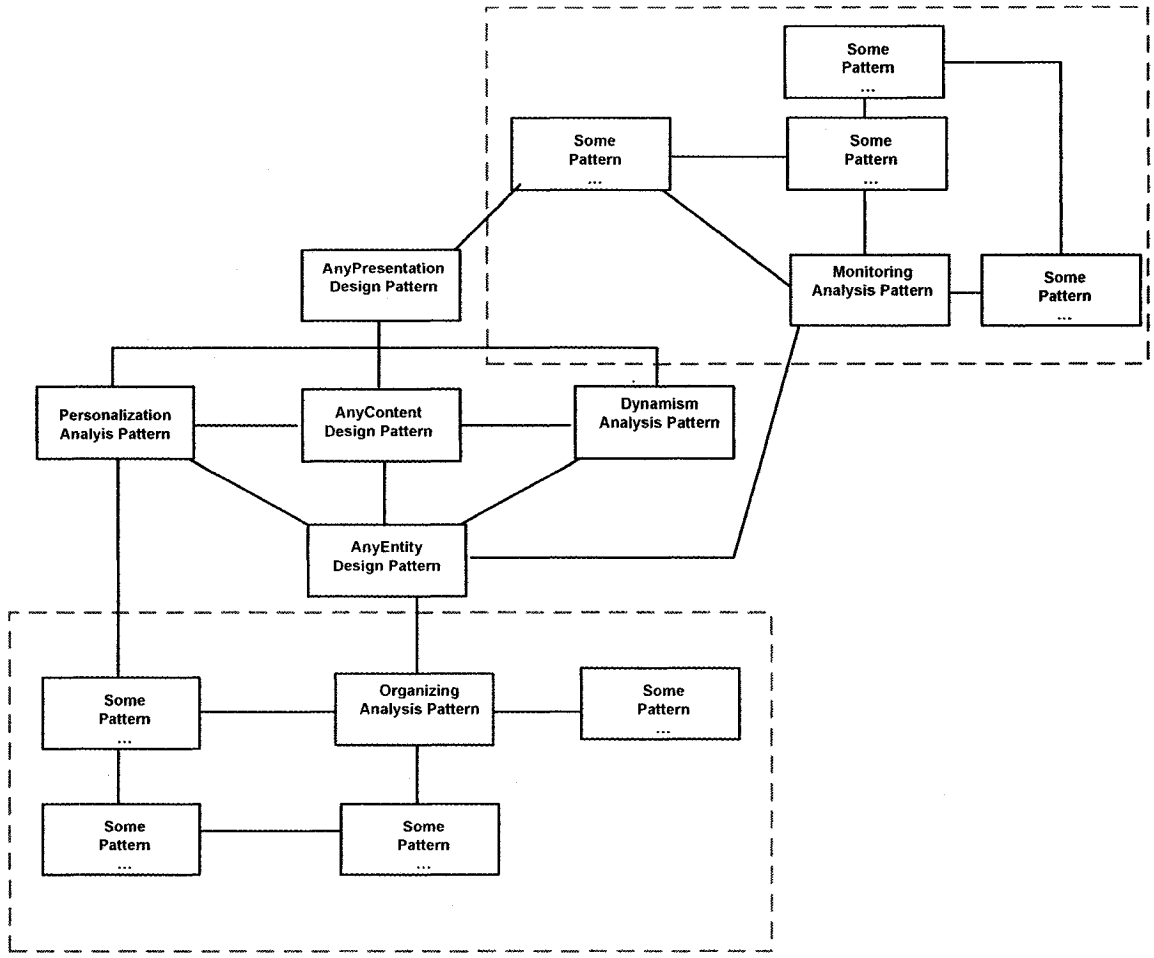

Figure 4.3. Representation of an extended UCME architecture.

A point to consider with regard to Figure 4.3 is that sometimes patterns are developed for domains other than the domain of focus. For example, AnyMedia design pattern might have been developed for the domain of wireless communications, but it might be of immense interest to be used in the content management domain. This is called connecting remote patterns. Figure 4.4 illustrates this concept at the domain level.

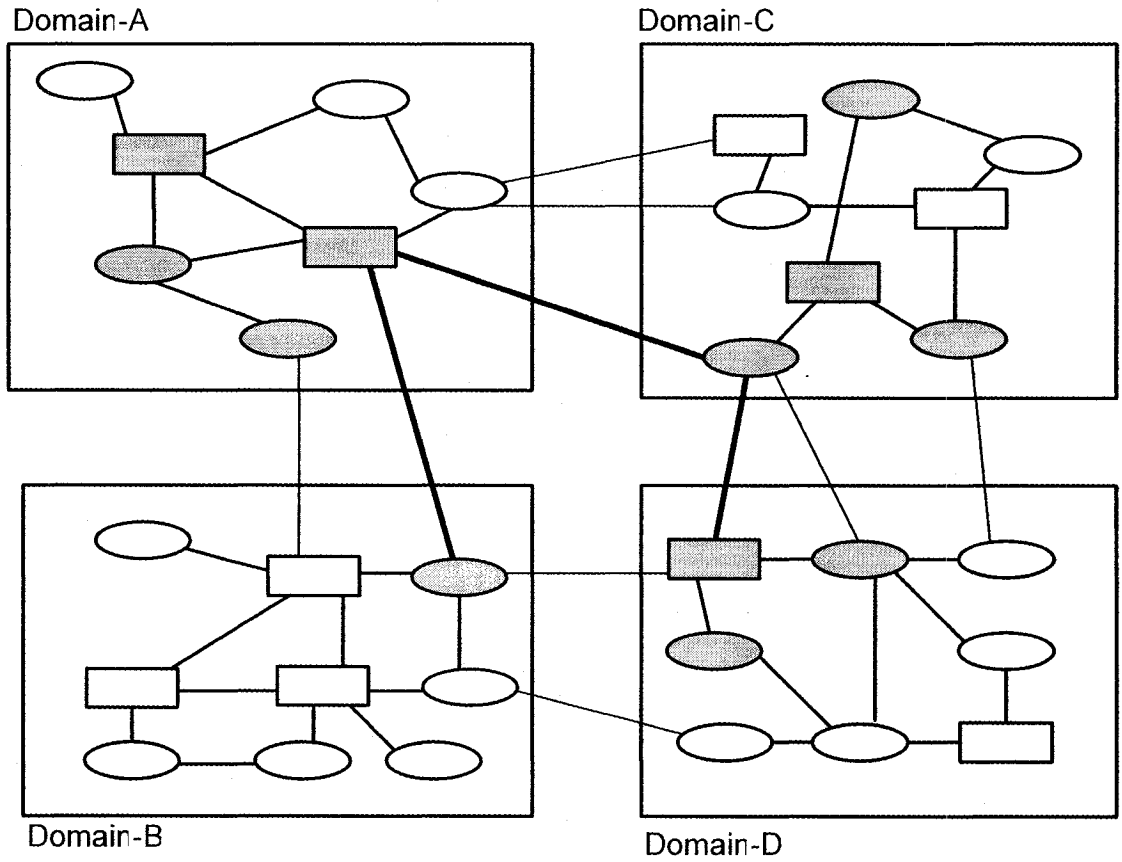

Figure 4.4. Representation of UCME architecture using a remote pattern.

In Figure 4.4, rectangles represent analysis patterns and ovals represent design patterns. A new architecture is formed in gray color by connecting analysis and design patterns across four remote domains.

4.2.2. Classifying Traditional CMSs

Content management systems could be classified into various categories depending on the perspective of studying them. Content management systems are quite complex systems composed of many features, various facilities, and a myriad of technologies. Therefore, classifying them could be a mind boggling exercise and a

confusing process. However, there are natural and logical methods for classifying and categorizing content management systems. Studying about various content management systems, several factors could be considered as the major principle for categorizing the traditional content management systems. For the rest of Section 4.2.2, each of the factors is briefly discussed and overviewed.

1. Domain of their applicability: This factor represents the domain to which content management systems are generally applicable. Examples for such domains are: web management domain, finance, education domain, government, and healthcare domains.

Domain of applicability could be considered a classifying factor only for the existing content management systems which are developed based on the traditional software development model. The enterprise content management systems which cover multiple domains would fall into multiple categories.

2. Employed technologies and capabilities: Content management systems could also be categorized by their technologies and capabilities. For example, certain content management systems such as the SCMS Flash introduced in Chapter Two are targeted towards managing Flash contents. Other capabilities of content management systems include security, management features, and the CGI-bin support.

3. Targeted technologies and audience: Content management systems could also be categorized by their audience. For example, certain content management systems only support the Perl programming language and are targeted towards Perl user communities. Others content management systems might only support Java. In contrast, some systems might only support PHP scripting or Python for their audience.

4. Marketing factors: Marketing factors including cost and types of licenses for using content management systems could categorize the systems. For example, content management systems with GPL licenses could be categorized separately from the content management systems that operate under commercial licenses. The free content management systems could be separated by the cost factor from the numerous other systems available.

4.2.3. Taxonomy for UCME

The aforementioned factors could be used for categorizing the existing content management systems as indicated by Gilbane Report ("Classification"). However, such factors would not be suitable factors for categorizing various stable architectures of UCME. This is due to the fact that the stable architectures are extensible in nature and can support various features per demand. They could also be deployed and used across various domains as UCME is domain-agnostic. Furthermore, due to its extensibility features, the UCME could be extended to support various technologies and target different markets.

The suggested method for classifying the stable UCME architecture is to categorize content management applications primarily based on the goals they support and secondarily based on the paths used to connect all such goals. This classification method would set all applications that support similar or the same goals in one category, and would allow further categorization of systems by the paths chosen to connect design EBTs.

4.3. Applications of UCME

Building software applications based on the proposed UCME would involve a series of different procedures. The first logical step is to gather some software requirements and determine all required functionalities for the content management system of interest. Then, based on the gathered requirements, the goals that satisfy the collected requirements must be recognized and included in the architecture design. The next step is to connect all goals and their empowering capabilities to achieve a full architecture. Next, the associated IOs for the BOs in architecture must be developed and hooked to the system. Finally, a system of the selected EBTs and BOs and all attached IOs would be built to generate a new application. The following steps, describe these procedures in abbreviated terms:

- Identify the requirements and goals of the application of interest
- Choose the EBTs and their empowering BOs from the patterns involved in UCME
- Connect the goals and capabilities to form a new architecture
- Attach all IOs to the BOs of the developed architecture
- Build the architecture and all IOs to generate the UCME application

For example, if the requirements and the recognized needs for creating a new CMS application based on the UCME only concern dynamism and personalization features for a website, both Dynamism and Personalization EBTs and their associated analysis and design patterns would be identified and connected just as illustrated in Figure 4.1 and Figure 4.2.

Then the IOs are connected to BOs to form an application. Figure 4.5 illustrates this example.

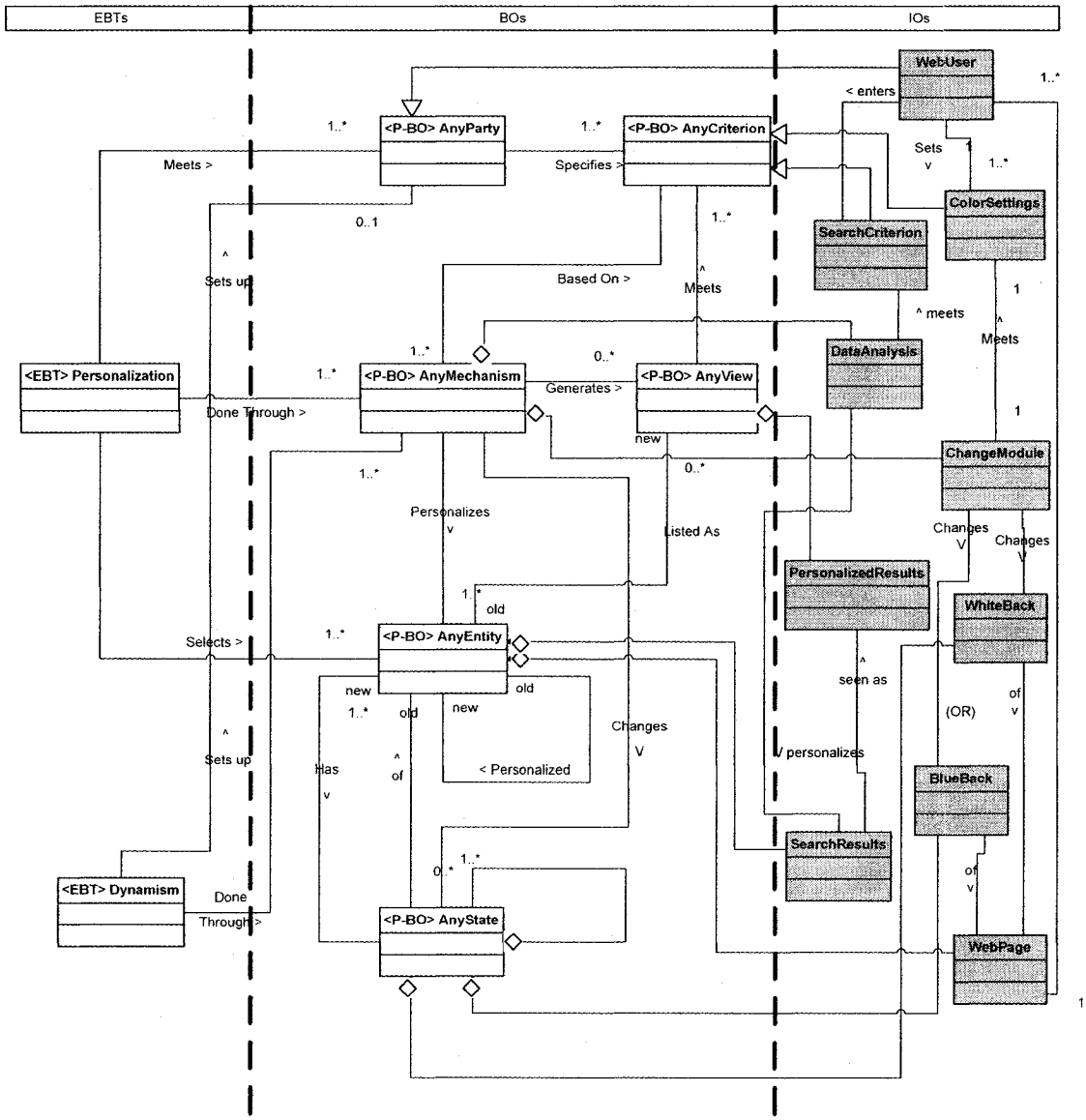

Figure 4.5. Dynamism and Personalization analysis patterns in UCME.

In Figure 4.5, the IOs are presented in gray and the rest of the engine is presented in white boxes with goals on the left most side and the BOs in the middle of the figure.

4.4. The UCME's Impacts

After introducing the UCME engine and the methods to develop various architectures and applications, Section 4.4 covers the impacts of using the UCME on various software related activities.

• **Technology:** The consequences of using applications that are based on the UCME are quite enormous in various fields of technology. Usually using new technologies in software or industrial developments depends on the availability of the infrastructures required by those technologies. UCME applications could easily cope and adjust with changes in technologies due to their full extensibility and flexibility characteristics. Therefore, there should not be issues or incompatibilities between the new and old techniques in the field of managing contents when the UCME is in use.

• **Business:** Software applications based on the UCME are extremely adaptable and flexible for deployment. This is a great phenomenon for businesses and corporations as such applications could be integrated with full adaptability into existing or new business flows with no fear or worries about possible incompatibilities between the new application and the existing legacy software or processes. In addition, all applications that are based on the UCME would buy the customers' acceptance due to the flexibilities that such systems inherit from the concept of stability used in developing the UCME.

Generally, applications developed based on the UCME have greater rate of return on investment. The cost of obtaining a software application that is developed based on the UCME does not include the cost of modules that remain unused by customers. When the architecture of an application is chosen through the software stability model, only the

needed goals and capabilities could be included in the application. Therefore, customers and businesses would not need to pay for what they do not use.

Businesses and software developers would benefit immensely from the flexibility offered by applications that use the UCME as they could stretch their applicability per demand. For example, users could add new IOs to make their application work with Oracle database or with any new system brought on board. This is due to scalability and extensibility characteristics of the UCME.

• **Education:** The UCME based software applications prove extremely effective in education domain, as they could be tuned to the needs of various audiences. These applications could be targeted for school and university management as well as students' records administration. Depending on the size and level of educational centers, the UCME based applications could be developed and customized to match the level and size of the center. For example, new students of a university –such as undergraduate students- would use a more simplified interface and features of an application based on the UCME, while more veteran students –such as graduate students- could use more complex and extensive services of the application. This is possible by offering different IOs to be attached to the core UCME.

All software applications which use the UCME could be tailored to specific needs of students and instructors such as for collecting assignment, running examinations, grading, and posting students grades. Flexibilities offered by applications that are based on the UCME would enable tailoring the applications to the needs of each instructor and the guidelines and teaching policies of an education center's staff.

• **Research:** Content management systems have many applications in various research projects and are usually very beneficial. This is due to the fact that many literatures, references, benchmarks, quotes, measurements, and notes must be kept accessible, categorized, indexed, versioned, published, and safe. A flexible content management system that could be tailored and tuned to the needs of various research fields and disciplines is of great value and would positively impact research processes in any field. The UCME based software applications are great candidates for such flexible content management systems. Such applications could be tailored to researchers' needs and are flexible to be extended to support interdisciplinary research across different fields and domains.

4.5. Conclusion

Chapter Four continued the study of the unified content management engine which was first introduced in Chapter Three. Following the extensive discussion about software design presented first in Chapter Three, the architecture of the UCME was presented in this chapter, and the concept of stability in software design was discussed and illustrated in more detail. Furthermore, methods of classifying the existing content management systems and the ways and means to categorize the UCME based software were presented. Then, a procedural description for developing software applications based on the UCME was presented along with a sample application. Finally, the impacts of using the UCME in various fields and domains were discussed in Chapter Four.

Chapter Five: Case Study

5.1. Abstract

Life is dynamic and vigorous! Four seasons come and pass every year; the leaves of the trees grow green, turn yellow, gold, and red before they shed in the brilliant white of the winter. The colors of nature change: dynamism at work, beauty on full display!

One of the major challenges for current Content Management Systems (CMS) as established in Chapter One and Chapter Two is to bring about dynamism and flexibility in their functionalities. In Chapter Five, the proposed solution, which was presented in Chapter Three and Chapter Four, is used in various scenarios of managing content. Specifically, using the concepts of dynamism and personalization in different contexts, this chapter documents scenario that verify the functionality of the offered solutions of this report. Furthermore, Chapter Five presents how to use the unified content management engine inside different applications with various IOs.

5.2. Introduction

Based on the results of Chapter Two's comparative study, this thesis identifies inflexibility as one of the main pitfalls of the existing content management systems. In the previous chapters, a solution for this problem was improvised and was integrated in a content management engine.

Personalization is a feature that enables users to personalize their contents based on their preferences and tastes. In this chapter, this feature of a unified content management engine will be discussed using a scenario.

In the context of managing contents, dynamism is defined as changing the representations of some contents in an automatic and orderly fashion. In this chapter, dynamism as a feature of the unified content management engine is discussed and presented.

Managing content is all about presenting, adding, deleting, modifying, publishing, versioning, and many other similar operations that could be applied over some content. Content could be anywhere and in any form or structure. The scenarios that are presented later in this chapter do not involve the mechanisms of retrieving and storing content because discussion of such procedures enter the realm of data mining and content storage which are in entirely separate domains. These should not be understood as limitations on the Unified Content Management Engine (UCME). In fact, one of the main advantages of using the Software Stability Model (SSM) for software design is that the SSM makes software scalable. The unified content management engine could always be extended with other features, and even easily communicate and collaborate with other stable engines such as the data visualization engine and the data mining engine. Figure 5.1 depicts a collaborative picture of various tasks that are closely associated with managing content. In Chapter Five, the focus is entirely on two features of the unified content management engine: dynamism and personalization.

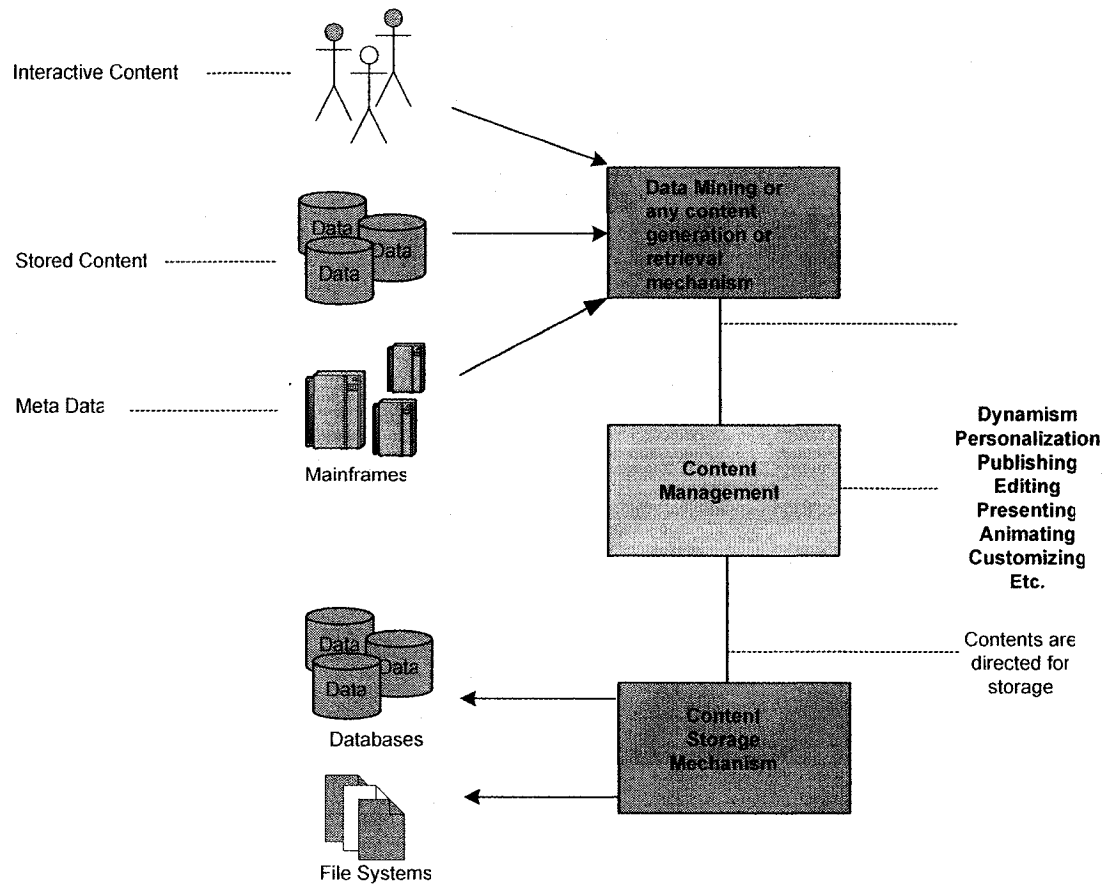

Figure 5.1. Content management in interaction with other content related activities.

The rest of the chapter illustrates and documents scenarios of managing content by making use of personalization and dynamism analysis patterns, which are parts of unified content management engine. Each scenario will use at least a use case to illustrate the design using sequence and class diagrams.

5.3. Problem

Section 5.3 demonstrates two separate and distinct scenarios in which two analysis patterns and their empowering design patterns are actively involved. The two

analysis patterns are the Personalization and Dynamism stable analysis patterns. Both analysis patterns are the major parts of the unified content management engine.

The scenarios are documented based on the SSM's documentation guidelines. In each scenario, one use case is presented.

Figure 5.2 depicts both Dynamism and Personalization analysis patterns at a cross point with their empowering BOs. Several IOs are also connected to the presented unified content management engine. In this chapter, two analysis patterns are studies in two scenarios.

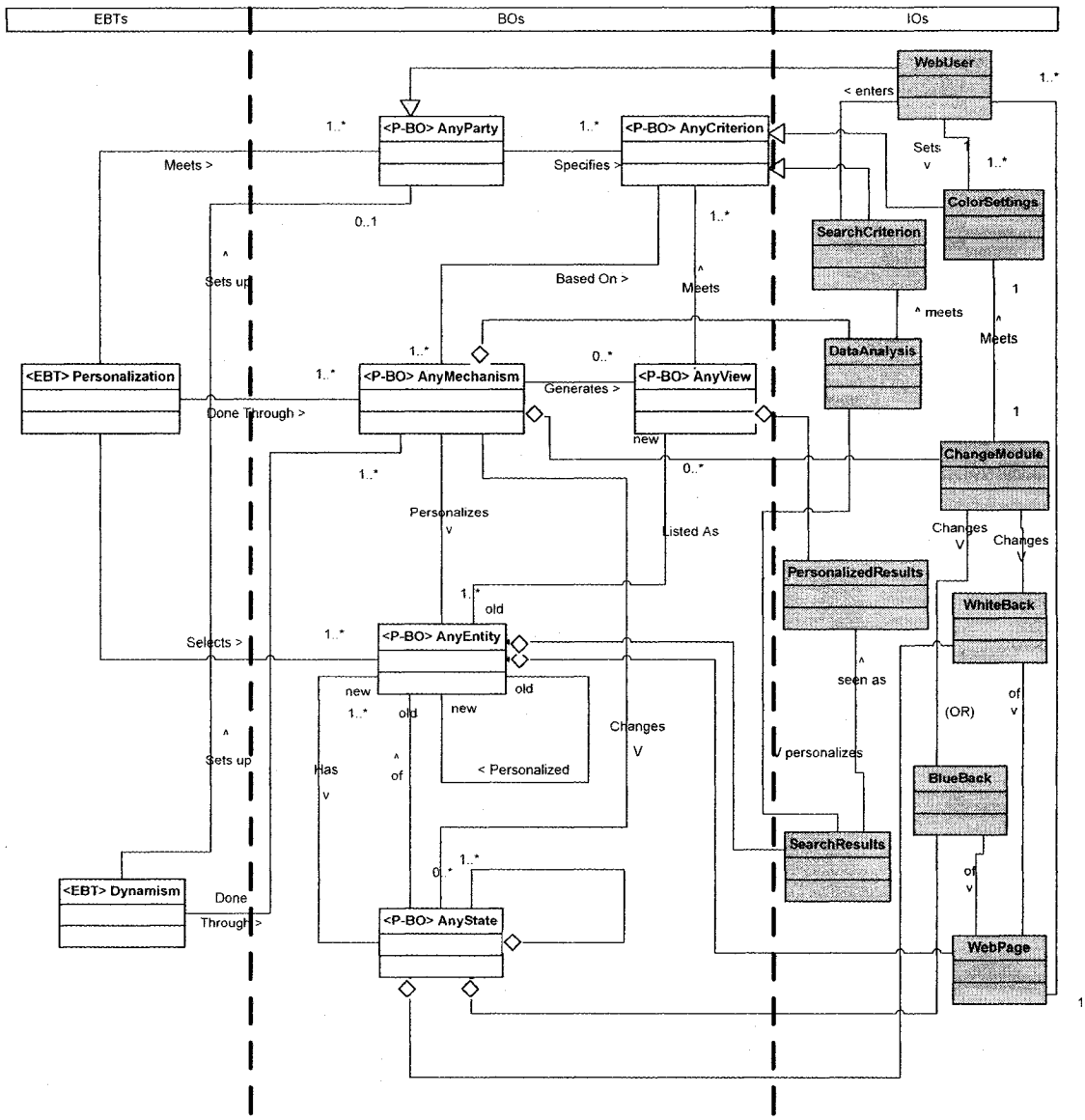

Figure 5.2. Dynamism and Personalization analysis patterns combined.

Figure 5.2 combines the Personalization and Dynamism analysis patterns in one single view and demonstrates how various IOs are attached to their related BOs. The left most section, which is separated from the rest of the design components with a dotted

line, is the region where all EBTs are located. The middle section between the two dotted lines is the BOs' region, and the right most section only contains the IOs.

In the following sections, two scenarios involving two EBTs and their associated BOs and IOs are presented: one for the Personalization analysis pattern and one for the Dynamism analysis pattern. Each scenario is described and further elaborated with a Unified Modeling Language (UML) use case and a UML sequence diagram. The first scenario is about personalization while the second scenario is about dynamism.

1. Scenario One: Personalizing Web Content

- **Scenario Identification:** 1
- **Scenario Title:** Personalizing a Web Search
- **Scenario Description:** In this scenario, contents of a web page will be shown to be personalized using the Personalization analysis pattern. An effective search for web content depends on several important factors like user's search history, geographical locality, language, and user's bookmarks that all are used implicitly in search result outputs. Furthermore, presenting search results on a web page could also involve personalization. For example, personalizing a search website to automatically load some selected news headlines based on a pre supplied criterion or personalizing a web page to list a selected set of entertainment links when the page is loaded is of general use and special interest. In this scenario, a targeted search using some implicit search criteria such as locality and user's browsing history is presented as a use case. Figure 5.3 represents a UML class diagram for the Personalization analysis pattern along with some IOs for the current scenario.

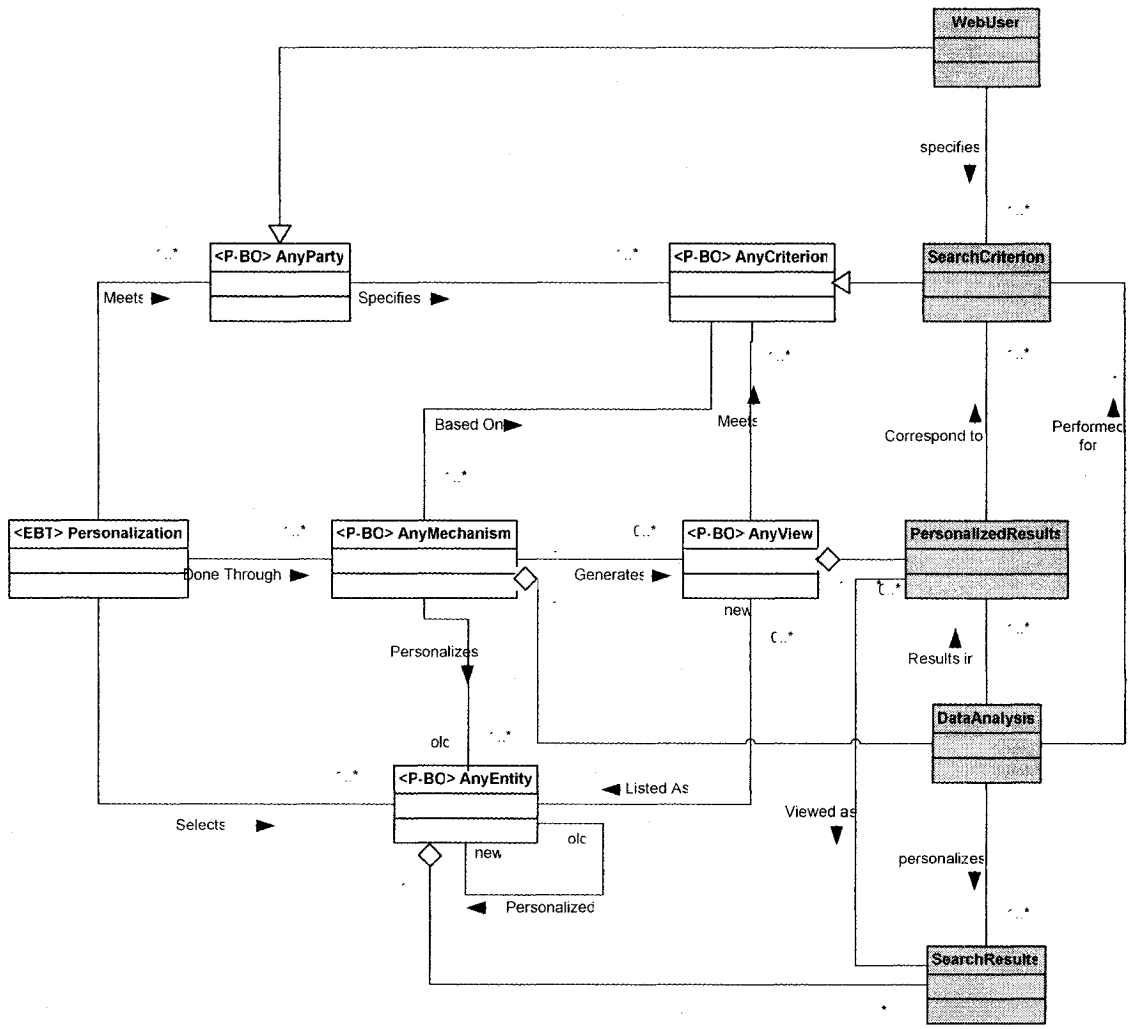

Figure 5.3. Personalization analysis pattern along with application IOs.

- **Scenario Patterns:**

Personalization Analysis Pattern

AnyParty Design pattern

AnyMechanism Design Pattern

AnyCriterion Design Pattern

AnyView Design Pattern

• **Challenges:** One of the major challenges in this scenario is to map appropriate IOs to corresponding BOs in a collaborative way so that pattern could reach and satisfy its goals. Another unique challenge is to specify a set of valid mechanisms for personalizing an entity. Finally, AnyMechanism class should be fine tuned to the details of AnyEntity.

• **IO Applications:**

WebUser

SearchCriterion

PersonalizedResults

DataAnalysis

SearchResults

• **Use Cases:**

Use Case ID: 1.0

Use Case Title: Personalized Search Results

Table 5.1 and Table 5.2 describe class components of the Personalization analysis pattern.

Table 5.1

Actors and Their Roles for Web Personalization Scenario

Actors	Roles
AnyParty	WebUser

Table 5.2

Classes in Personalization Analysis Pattern and Their Descriptions

Classes	Type	Attributes	Operations
Personalization	EBT	• Content	• initiate()
AnyEntity	BO	• cause • effect • type	• characterizePrototype()
AnyParty	BO	• name • role • identity	• chooseOption() • selectChoices() • provide()
AnyMechanism	BO	• type • attribute • context	• generateView()
AnyCriterion	BO	• quality • value description	• specifyCharacterstic()
AnyView	BO	• viewDescription • size • viewLink • style	• illustrate()
WebUser	IO	• interest • areaOfWork	• statePrefersnce() • getPersonalizedResult()
SearchCriterion	IO	• hasPreference	• query() • searchParameter()
SearchResult	IO	• techniques • displayType	• responseToCriterion()
UserProfiling	IO	• noOfOutput • speed	• mechanosmForResult()
DataAnalysis	IO	• model • type • make	• mechanosmForResult()
PersonalizedResult	IO	• characteristic	• getPersonalizedWeb()

Enduring Business Themes (EBT): Personalization

Business Objects (BO): AnyParty, AnyMechanism, AnyCriterion, AnyView, AnyEntity

Industrial Objects (IO): WebUser, SearchCriterion, SearchResult, DataAnalysis, UserProfiling, PersonalizedResult

Description of the Use Case:

1. A WebUser (AnyParty) defines a SearchCriterion (AnyCriterion) and searches the web using a search engine.

2. WebUser (AnyParty) requests to personalize his SearchResult (AnyEntity).

3. Personalization (EBT) is applied over user's SearchResult (AnyEntity).

4. DataAnalysis (AnyMechanism) and UserProfiling (AnyMechanism) use the SearchResult (AnyEntity) and the SearchCriterion (AnyCriterion) to create a PersonalizedResult (AnyView).

5. The PersonalizedResult (AnyView) is made visible to the WebUser (AnyParty) according to the preferences that were set using the SearchCriterion (AnyCriterion).

6. Finally, Personalization (EBT) returns the SearchResult (AnyEntity) to the WebUser (AnyParty).

Sequence Diagram:

Figure 5.4 depicts the sequence diagram for this scenario.

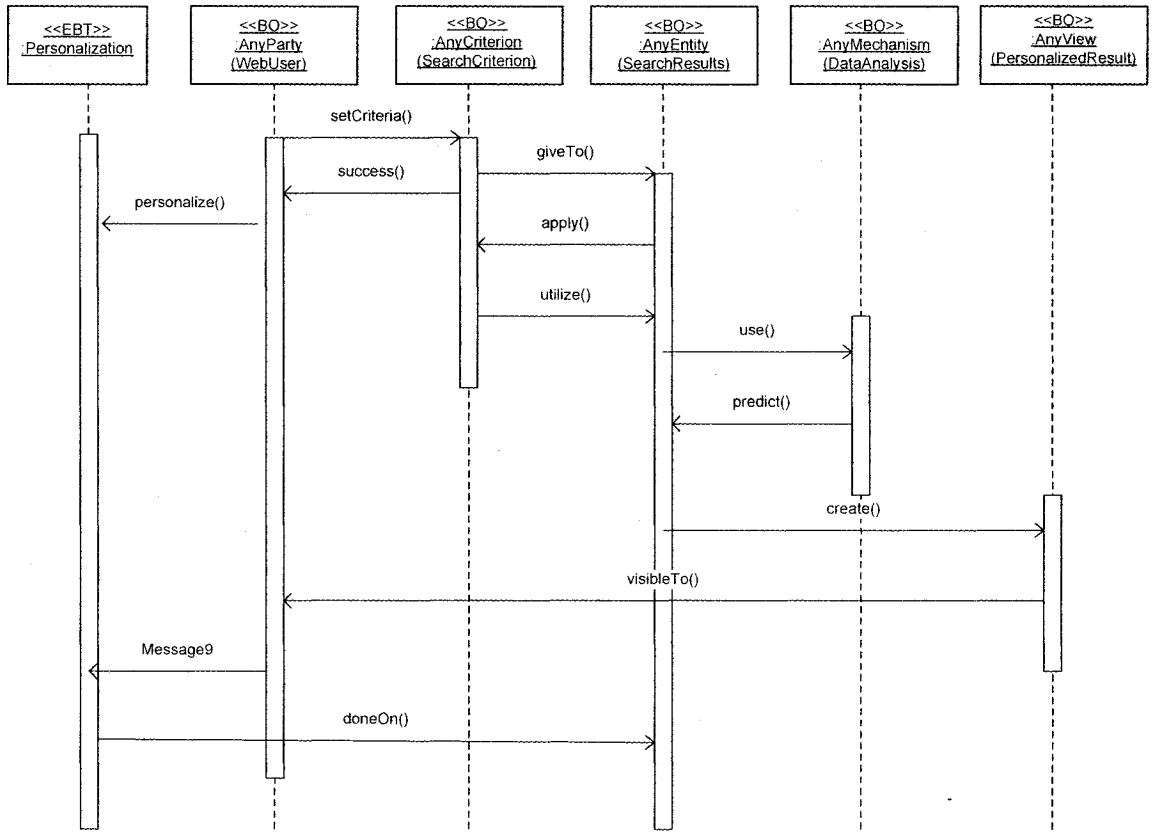

Figure 5.4. Sequence diagram for personalized search results.

2. Scenario Two: Website Color Change

- **Scenario Identification:** 2
- **Scenario Title:** Website Color Change
- **Scenario Description:** A common scenario in web development domain is the case where users desire to dynamically update their web page by changing its theme colors, updating various content components periodically, and showing certain contents based on some time factors.

The following documented scenario illustrates how the concept of dynamism could be used in a limited, yet simple way to change a web page's background color. In this scenario, a user accesses a website and would attempt to make the page dynamically changing by setting certain criteria to dynamically change the web page's background color every ten minutes. In this scenario, dynamism is the phenomenon of change in the background color of a web page. The WebUser represents AnyParty, and the code module that dynamically changes the web page's background color represents AnyMechanism. The WebPage represents AnyEntity. Finally, the AnyState represents various states of the WebPage in this scenario. Figure 5.5 shows the class diagram for the Dynamism analysis pattern along with the related IOs.

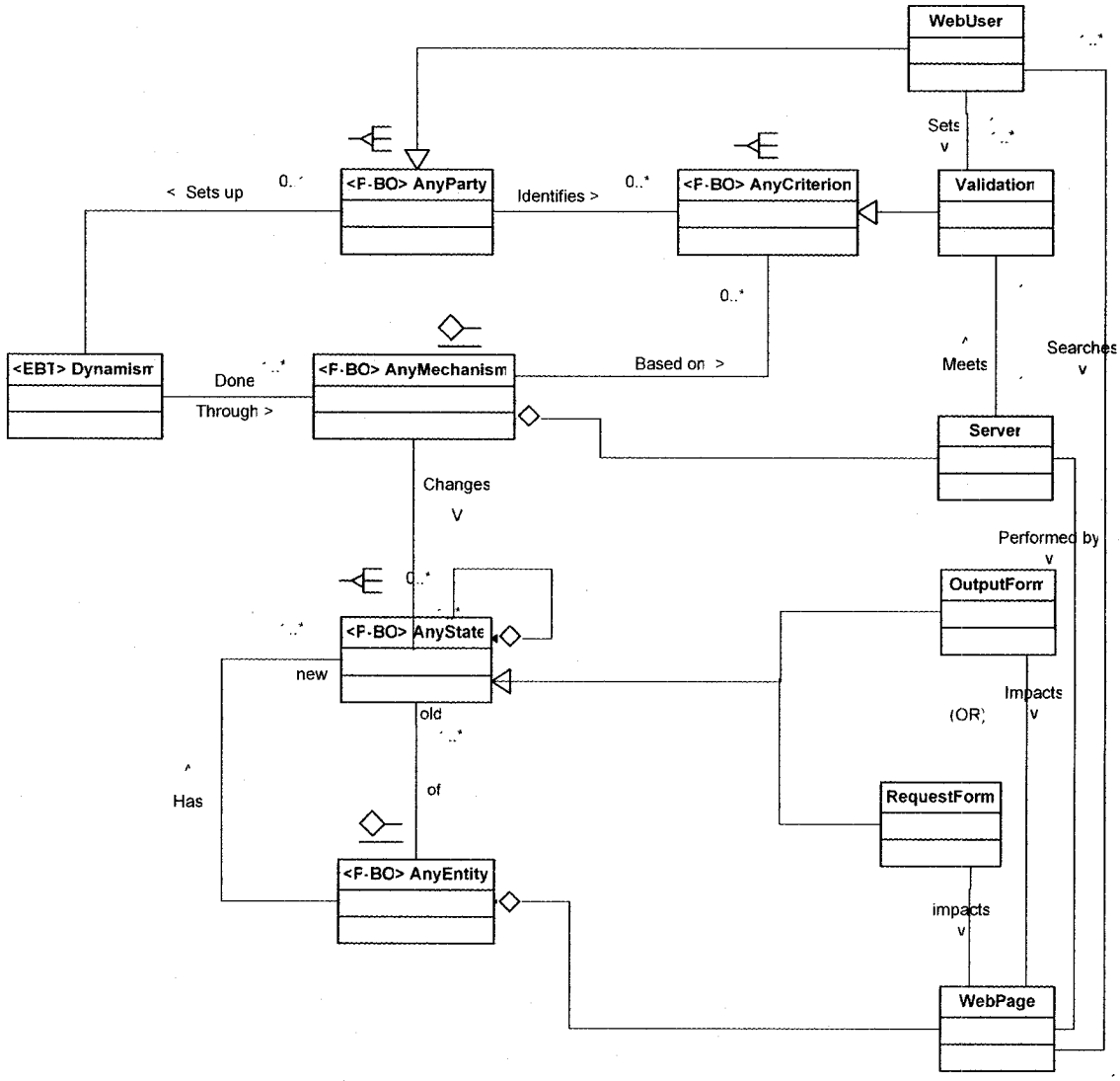

Figure 5.5. Dynamism analysis pattern along with web IOs.

- **Scenario Pattern:**

Dynamism Analysis Pattern

AnyEntity Design Pattern

AnyCriterion DesignPattern

AnyView Design Pattern

AnyParty Design Pattern

AnyMechanism Design Pattern

- **Challenges:**

One of the challenges in this scenario is choosing a valid criterion for AnyMechanism. Any failure to choose a valid criterion would cause the Dynamism to stop or fail as AnyMechanism would stop due to using an invalid criterion.

- **IO Applications:**

WebUser

ColorSettings

WhiteBack

BlueBack

ChangeModule

WebPage

- **Use Cases:**

Use Case ID: 1.0

Use Case Title: Bringing Dynamism to a Web Page.

Table 5.3 and Table 5.4 describe class components of the Dynamism.

Table 5.3

Actors and Their Roles in Dynamism Analysis Pattern

Actors	Roles
AnyParty	WebUser

Table 5.4

Classes Involved in Dynamism Analysis Pattern

Classes	Type	Attributes	Operations
Dynamism	EBT	• Process	• change()
AnyParty	BO	• name • identity • phoneNumber	• initiate()
AnyMechanism	BO	• purpose • privilege • step • criteria	• doDynamism() • checkCriteria() • reportResult()
AnyEntity	BO	• type • name • characteristic	• create() • updateState()
AnyCriterion	BO	• criteria • agreement	• setCriteria() • reportCriteria()
AnyState	BO	• intialState • finalState	• changeState() • showState() • report() • updateEntity()
WebUser	IO	• purpose • parameter	• addCriterion() • startDynamism()
ColorSettings	IO	• validationRule	• validate() • addColor() • removeColor() • currentColor()
ChangeModule	IO	• configuration	• actsOn() • processChange()
WebPage	IO	• type • use • information	• sendRequest() • refresh()
BlueBack	IO	• color • name	• takeUserInformation() • showInformation()
WhiteBack	IO	• color • name	• takeUserInformation() • showInformation()

Enduring Business Themes (EBT): Dynamism

Business Objects (BO):

AnyParty

AnyMechanism

AnyCriterion

AnyEntity

AnyState

Industrial Objects (IO):

WebUser

ColorSettings

ChangeModule

WebPage

BlueBack

WhiteBack

Description of the Use Case:

1. A WebUser (AnyParty) chooses various colors and a time interval as values for ColorSettings (AnyCriterion). The ColorSettings class defines all possible background colors that a WebPage (AnyEntity) could have. Color checking is done in this stage, and unacceptable colors will be rejected. The success will be reported to a WebUser (AnyParty).

2. When the WebUser (AnyParty) saves the ColorSettings (AnyCriterion), the request to dynamically update the WebPage (AnyEntity) is sent to Dynamism.

3. As soon as Dynamism is initiated, a ChangeModule (AnyMechanism) begins to dynamically change WebPage's background color based on the values of ColorSettings (AnyCriterion).

4. The ChangeModule (AnyMechanism) would then update the color of the WebPage (AnyEntity) based on the time interval specified in ColorSettings (AnyCriterion).

5. Then, The ChangeModule (AnyMechanism) changes the state of the WebPage (AnyEntity) by switching the background color to WhiteBack (AnyState) or BlueBack (AnyState).

6. Finally Dynamism returns the WebPage (AnyEntity) to the WebUser (AnyParty).

Sequence Diagram:

The sequence diagram for dynamic updates of a web page is shown in Figure 5.6.

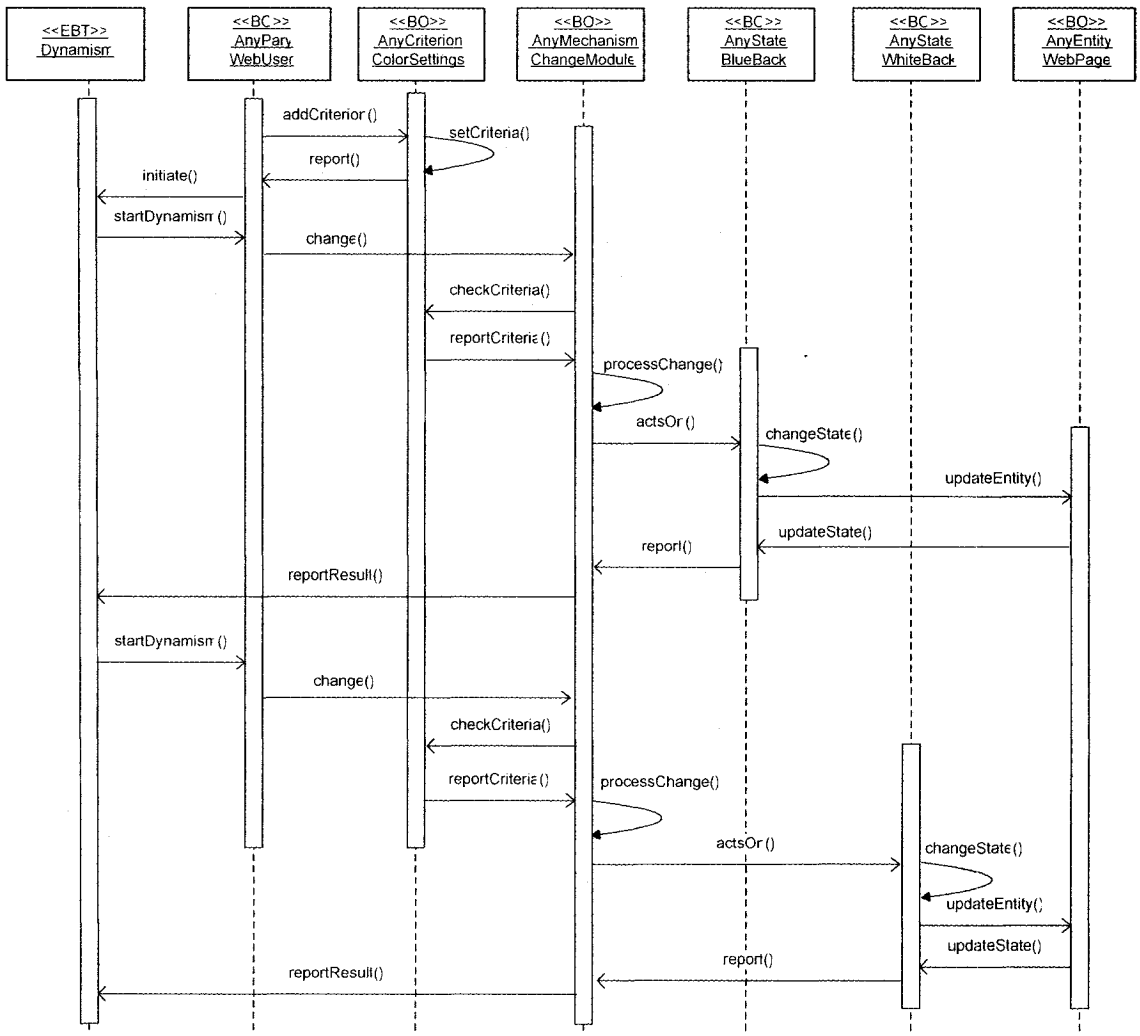

Figure 5.6. Sequence diagram for brining dynamism to a web page.

5.4. Conclusion

In Chapter Five, the Personalization and Dynamism analysis patterns were presented in two separate scenarios. Each scenario was described and documented by using use cases and sequence diagrams. Moreover, class diagrams for each scenario along with a class diagram for both scenarios combined were presented. Chapter Five

also discussed the functionalities of the Personalization and Dynamism analysis patterns as two components of the unified content management engine.

Chapter Six: Analysis and Major Findings

6.1. Introduction

This report documents research which is mainly aimed at providing some concrete and robust solutions to major problems that surround the existing content management systems. In this research effort, a comparative study was performed to detect, identify, and illustrate a host of major problems that Content Management Systems (CMS) are currently challenged with. Furthermore, building on the existing and available bases, the fundamental concept of software stability was introduced and explained in length. It was also illustrated how the Software Stability Model (SSM) can be used to solve the identified and listed problems in Chapter One and Chapter Two. The solutions introduced in Chapter Three and Chapter Four provided a solid base upon which various content management system applications could be built, and a case study was presented later in Chapter Five. In this chapter, the major findings of this research are listed, and an analysis is performed on the achievements of this research work.

6.2. Major Findings and Achievements

In Section 6.2, the major achievements and accomplishments of this research study are listed, and major findings of this research are analyzed in Section 6.3.

Following is the list of the major achievements of this study:

- Some of the major problems with the existing content management systems were identified through an elaborate comparative study. In the comparative study, six major content management environments or tools were reviewed.

- The concept of software stability model was described in length and illustrated by various diagrams and illustrations.

- Many goals and capabilities for a stable content management engine in terms of Enduring Business Themes (EBT) and Business Objects (BO) were identified and described. This task required a great amount of exploration and discussion as to identify the true goals or objectives of the system and the capabilities which enable the goals in various applications. One of the major challenges in this task was identifying the most apt and the best fitting set of goals for the system among many similar such possibilities.

- One other major contribution of this work, after recognizing the ultimate goals and their associated capabilities, is designing and crafting the analysis patterns based solely on the software stability model. This was certainly a difficult task that involved studying various cases and later looking into many scenarios to capture all possible design requirements. In this report, two major analysis patterns are presented and described. Chapter Three explores the two analysis patterns mentioned.

- Another major achievement in this research is the development of design patterns based on the identified BOs. Similar to design of the analysis patterns, many scenarios were considered and probed to ensure the completeness of the design. In case of problems, a revision work was required to identify the issues in the previous steps such as

possible missing goals and capabilities and addressing the shortcomings in crafting the analysis patterns. Three major design patterns are presented and described in this thesis work. Chapter Three explores the design patterns.

- In this thesis, a knowledge map, which describes how various components of the captured knowledge are related in terms of EBTs and BOs, was introduced for content management systems.

- A new classification is proposed to map the goals and capabilities that are captured for applications of the content management system. The knowledge map can map the goals to all required capabilities. Therefore, depending on the goals for which an application is being developed, the knowledge map would determine all required capabilities and possibly other related goals. This would enable developers to construct a new software application for the specific goals in mind.

- An efficient content management engine is proposed in this research as a solution to the prior list of pitfalls associated with the existing content management systems. The proposed solution would fully resolve the listed problems. The beauty of this work rests over the stability of the solutions that enables them to be used in any domain of managing contents. Moreover, the proposed solution further reduces the maintenance and management of the engine as the engine could adapt to future changes in technology.

- The introduced analysis and design patterns for the content management engine could be used as part of other engines or simply used as stand-alone patterns as part of any ad

hoc design. The introduced analysis patterns are documented as part of this report's appendices.

• This report is not just limited to proposing conceptual designs alone. Some of the patterns presented in this thesis are coded and implemented for demonstration purposes. There is a dedicated appendix, which documents sample code of these patterns. This would be useful for learning and understanding purposes, the documented code will help to get familiar to implementation of stable patterns.

• Scenarios are implemented to use and adapt the coded patterns in various contexts for demonstration purposes. The scenarios exercise the developed patterns in different ways and invoke their procedures and capabilities to show that the design can satisfy users' needs.

6.3. Analysis

This thesis report is very rich in achievements and findings. The achievements in this report could be used entirely like the content management engine or could be used partly like the documented patterns as part of other designs.

The achieved engine design has superior architecture when compared to existing content management software applications as it could be used in any scenario of managing content. The achieved engine when fully completed and implemented would be highly flexible, efficiently adaptable, fully extensible, and could be integrated with other stable applications.

The patterns introduced and developed in this work could also be used individually in other designs. In fact, all patterns developed through the software stability model have the capacity of being connected to other relevant stable patterns. Therefore, the patterns developed in this work could be used in many other scenarios and for various purposes. In addition to the patterns, the implementations could also be used in modular forms, which greatly contribute to software reuse.

Finally, the presented work could be integrated with and used in conjunction with other stable engines or applications such as data mining or data visualizing applications. This would ultimately increase accessibility of content across various domains.

6.4. Conclusion

With numerous contributions from this work and many possibilities to use the results of this work, the values and benefits added by this research are certainly considerable and voluminous.

Chapter Seven: Future Work and Conclusions

7.1. Introduction

The contributions and major findings of this research are described and listed in Chapter Six. Based on the listed findings, there are several tasks that still need to be done for creating a fully functional content management software application. The future developments that could lead to this goal are presented in Section 7.2. Finally, Section 7.3 reviews the tasks that were undertaken in this research and lists the achievements.

7.2. Future Developments

The following tasks are recognized to be undertaken to further use the design suggested in this thesis report:

- One of the main tasks to be undertaken is to verify the proposed engine's applicability in other domains. The engine shall be tested and analyzed more comprehensively in as many scenarios as possible to ensure its soundness and full adaption to various users' environments. There will be fewer incidences of failures in adaption to users' environments for more accurate designs. In case of any problem in adaption, the issue must be traced back to the original design and analysis patterns to ensure the associated knowledge is captured.

- For the purposes of demonstrations, this research implements selected design components or patterns. Since the domain of managing content is vast, it is outside of this thesis' capacity and domain to implement all patterns for all goals in design.

Therefore, to use certain goals and capabilities of the engine, the patterns associated to the goals must get implemented as a viable future task.

- Not all functions in the implemented patterns are fully carried out. The interfaces are fully in place, but the functions that are not used for demonstration purposes are not fully implemented until now. Therefore, another future task would be to complete the implementation of all features for implemented patterns.

- Another important task is to expand the capacities of the engine and complete it by implementing the remaining identified goals and capabilities. Another future task would be to generate and implement all patterns that increase the software's capabilities such as visualization, and security modules. The current engine is somewhat limited as sections of the engine are already constructed but other sections are not developed.

- Another future task is to expand other capacities of the engine and use them by developing some Industrial Objects (IO) for the existing Business Objects (BO). For example, support for system's communication with various database servers such as Oracle, My SQL, and support for interacting with web servers such as Apache must be placed in.

- The existing developed patterns could be used in any relevant scenario. Another future task is to offer the design patterns as stand-alone patterns that could be used in users' designs. The engine should get equipped with facilities that allow novice users to integrate it easily with their application-specific software components. The engine should also be equipped with grammar that could be used by advanced users such as

programmers and developers to integrate the engine with their application-specific software components.

- Preparations must be made to turn the developed engine into a black box, so users could use it without the need to know how the engine is designed and implemented.

- Documents and guidelines must be composed to show users how to use the engine in simple or advanced modes. The documents need to describe all public methods and interfaces of the engine.

7.3. Conclusion

Current content management systems struggle with many problems that are inherited from the problems in their designs. In this report, the comparative study that is carried out in Chapter Two shows that many problems that current content management systems are struggling with have deep roots in their designs. Inflexibility, lack of full extensibility, and limited adaptability are just examples of the problems that originate from design of the existing content management system. This thesis uses the software stability model as described in Chapter One to overcome such problems. More similar problems are recognized and listed in Chapter One and Chapter Two. The problems are shown to be solved by capturing and separating the application knowledge from the rest of the application logic and offering the application knowledge as an engine that could be placed and used in any relevant users' scenarios. The proposed engine could be used in any domain of managing content because it captures all the relevant knowledge through following the software stability model guidelines described in Chapter One. Essentially,

the application knowledge has to do with the ultimate goals and objectives of an application.

Content management systems are used to satisfy various goals for managing various contents such as organizing, publishing, or controlling the content. When all goals are identified and the capabilities that empower the goals are listed, then the goals and capabilities are mapped to create a knowledge map. Then, using the knowledge map, a stable engine could be created. Since the captured knowledge would represent all the identified goals, then the engine could satisfy all users' needs and goals. Therefore, the engine is bound to satisfy all users' goals related to the task of managing content.

The content management engine is presented in Chapter Three and Chapter Four as a solution to the problems listed in this thesis. There is a case study in Chapter Five that discusses this topic more specifically. The software code and the documentations for some of the patterns used in this thesis are presented in the appendices sections of this report.

Finally, the major achievements of this research and the future development work that could lead to building a fully functional content management system are presented in Chapter Six and Chapter Seven.

Works Cited

"2007 Office System Pricing and Upgrade Information" OfficeOnline. August 2007. Microsoft Corporation. 17 August 2007 <http://office.microsoft.com/en-us/products/FX101754511033.aspx>

"About Drupal." Drupal. Vers. 5.3. 2007. 25 October 2007 < http://drupal.org/about>

The Apache Cocoon Project. 2007. The Apache Software Foundation. 20 August 2007 <http://cocoon.apache.org/2.1/features.html>

Apache Lenya. 2007. The Apache Software Foundation. 20 August 2007 <http://lenya.apache.org/>

"Apache Lenya 1.2 Install Instructions." Apache Lenya. Vers. 1.2. July 2007. 25 August 2007 <http://lenya.apache.org/docs/1_2_x/installation/source_version.html>

"Apache Lenya – License." Apache Lenya. 2007. The Apache Software Foundation. 20 August 2007 <http://lenya.apache.org/license.html>

"Apache Lenya – Open Source Content Management." Apache Lenya. 2007. The Apache Software Foundation. 20 August 2007 <http://lenya.apache.org/#-N1011F>

"Applicant Tracking." Allofe Solutions. 20 August 2007 <http://www.allofe.com/gen/corp_generated_pages/Product_Home_m255.html>

"Benchmarking PostNuke, Xaraya, Drupal, Joomla and e107." EZOSHosting 28 August 2007 <http://sascha.us/uploads/cms-benchmarks.pdf>

"The Classification and Evaluation of Content Management Systems" The Gilbane Report 11.2 (2003): 1-13 25 September 2007 <http://gilbane.com/artpdf/GR11.2.pdf >

"CMS Matrix.2007. The Compare Stuff Network. 15 August 2007 <http://www.cmsmatrix.org>

"Corporate, Government & Health Care." Allofe Solutions. 20 August 2007 <http://www.allofe.com/gen/corp_generated_pages/Business_m268.html>

"Enterprise Content Management." WebDot: The Open Text Web Solution Group. 21 August 2007 <http://www.reddot.com/products_enterprise_content_management.htm>

Fayad, M.E., "How to Deal with Software Stability." Communications of the ACM 45.4 (2002): 109-112.

Fayad, M.E., and A. Altman, "Introduction to Software Stability." Communications of the ACM 44.9 (2001): 95-98.

Fayad, M. E., "Accomplishing Software Stability." Communications of the ACM 45.1 (2002): 95-98.

Fayad, M.E., and Sanchez, H.A., "Knowledge Maps = Stable Pattern Languages"

"Features and Modules." Bitrix Site Manager. 2007. 21 August 2007. <http://www.bitrixsoft.com/sitemanager/modules/>

Hamza, H., and Fayad, M.E., "A Pattern Language for Building Stable Analysis Patterns." 9th Conference on Pattern Language of Programs (PLoP 02). Illinois, USA, Sep. 2002.

Hamza, H., and M.E., "Building Stable Analysis Patterns Using Software Stability." 4th European GCSE Young Researchers Workshop 2002 (GCSE/NoDEYRW 2002). Erfurt, Germany, Oct. 2002.

Hamza, H., A Foundation For Building Stable Analysis Patterns. Diss. University of Nebraska-Lincoln, 2002.

"Information Age." 04 September 2007
<http://en.wikipedia.org/wiki/Information_Age>

"Installation." Plone. 18 Sep. 2007. 15 October 2007 < http://plone.org/products/plone-starter/documentation/how-to/installation>

"Installation Guide." Joomla. 18 August 2007
<http://dev.joomla.org/content/view/2013/93/>

"Install Drupal 5.x" Drupal. Vers. 5.x. 2007. 25 October 2007 <http://drupal.org/getting-started/5/install>

"Install Office SharePoint Server 2007 on a stand-alone computer" Microsoft TechNet. Oct. 2007 <http://technet2.microsoft.com/Office/en-us/library/bd99c3a9-0333-4c1c-9793-a145769e48e61033.mspx?mfr=true>

Joomla. Vers. 1.0. August 2007. 16 August 2007 <http://www.joomla.org/>

"Microsoft Enterprise Content Management" OfficeOnline. August 2007. Microsoft Corporation. 17 August 2007 <http://office.microsoft.com/en-us/sharepointserver/HA101747881033.aspx>

"Microsoft Office SharePoint 2007 Server and Related Technologies Pricing" OfficeOnline. August 2007. Microsoft Corporation. 17 August 2007 <http://office.microsoft.com/en-us/sharepointserver/FX102176831033.aspx>

"Personalization." Wikipedia. 18 October 2007
<http://en.wikipedia.org/wiki/Personalization>

Plone. Vers. 3.0. 2007. 15 October 2007 <http://plone.org/>

"Replication (Computer Science)." Wikipedia. 22 August 2007
<http://en.wikipedia.org/wiki/Database_replication>

"Rewrite Engine." Wikipedia. 22 August 2007
<http://en.wikipedia.org/wiki/URL_rewriting>

SCMS Flash Content Management. 2007. 18 August 2007
<http://www.flash-content-management.de/>

"What Is Archetypes" Plone. 2007. Plone Foundation. 16 August 2007
<http://plone.org/documentation/manual/archetypes-developer-manual/what-is-archetypes/what-is-archetypes>

"What Is Content Management?" The Gilbane Report 8.8 (2000): 1-9 15 August 2007
<http://gilbane.com/artpdf/GR8.8.pdf>

"What Is Joomla?" Joomla Vers. 1.0. August 2007. 17 August 2007
<http://www.joomla.org/content/view/12/26/>

"Zope." Wikipedia. 17 August 2007 < http://en.wikipedia.org/Zope >

Appendix A: The Personalization Stable Analysis Pattern

A.1. Introduction

"Personalization is tailoring or streamlining a consumer product, electronic or written medium to a user, based on personal details or characteristics they provide" as defined by the Wikipedia ("Personalization"). Since the function of personalization is primarily dependent on the context in which personalization is being applied, the traditional approaches to software design will not result in a stable and reusable model. However, by using the Software Stability Model (SSM), the concept of personalization can be applied and used in any context using a single model. The software stability model requires creation of a knowledge map for a design by identifying its underlying Enduring Business Themes (EBT) and Business Objects (BO). Then, by hooking the application-specific Industrial Objects (IO) to the BOs, the model can be applied to any application domain. The resulting Personalization analysis pattern would be quite stable, reusable, extendable, and highly adaptable. Therefore, any number of applications could be built using this common model. The Personalization analysis pattern attempts to capture the core knowledge of the concept of personalization which is common to many applications by probing and analyzing various application domains and scenarios.

The overall objective of Appendix A is to document a stability model for the concept of personalization by creating the knowledge map of personalization. This knowledge map or core knowledge can then serve as a building block for modeling different applications in diverse domains. The rest of this appendix presents the

Personalization analysis pattern: a stable analysis pattern based on the software stability model.

A.2. Personalization Analysis Pattern Document

The pattern presented in Section A.2 utilizes the software stability model to explain the concept of Personalization. Figure A.1 depicts the class diagram for Personalization analysis pattern.

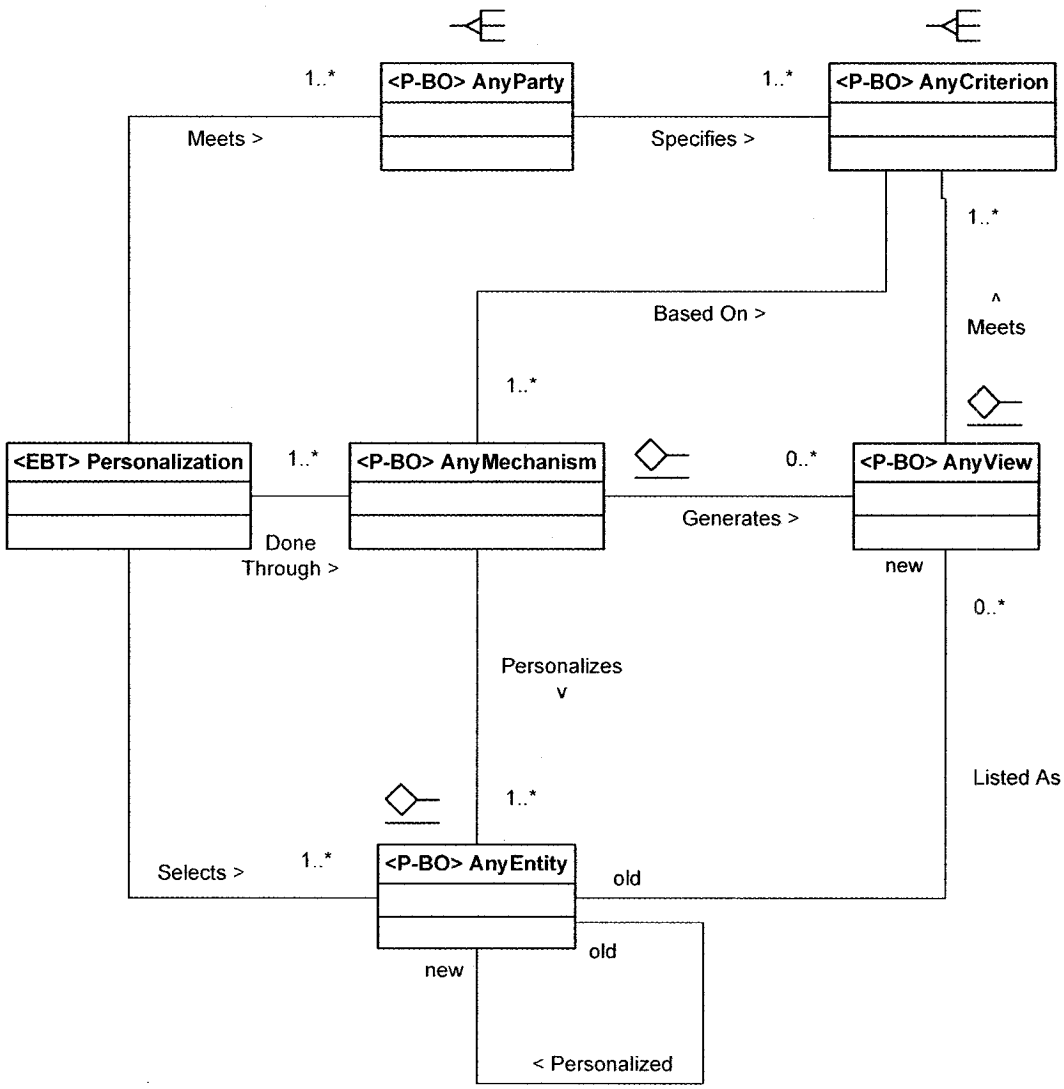

Figure A.1. Class diagram for Personalization analysis pattern.

The rest of Section A.2 documents the Personalization analysis pattern based on the software stability model's documentation template.

Pattern Structure and Participants

The participants of the Personalization pattern are presented below.

Classes

Personalization: This class represents the concept of personalization. It is an Enduring Business Theme, which presents an enduring concept that discloses relevant information based on users' specifications.

Patterns

AnyParty: This class represents any person, individual, organization, or a group with specific preferences who is interested in personalizing some entities.

AnyCriterion: This class denotes the factors necessary for any personalization process, which are determined by any person or organization. They may be gathered through interacting with AnyParty or from the past activities of AnyParty.

AnyEntity: It represents the description of some contents based on the utilized criteria that are specified by AnyParty.

AnyMechanism: This class represents the various techniques employed to carry out the process of personalization for AnyParty.

AnyView: This class represents a view of a personalized model generated by AnyMechanism.

Class Diagram Description

The class diagram presented in Figure A.1 provides a visual illustration of all classes involved in the proposed model along with their relationships. Description of the class diagram is as below:

1. Personalization is the EBT of this pattern.

2. Personalization (EBT) is applied on AnyEntity (BO) to represent some content. A view of AnyEntity is provided to AnyParty (BO).

3. AnyParty (BO) specifies some criteria using AnyCriterion (BO) class.

4. AnyMechanism (BO) uses AnyEntity (BO) to create AnyView (BO) of the personalized entity.

5. AnyView (BO) is created by AnyMechanism (BO), which ultimately would become a part of AnyEntity (BO).

CRC Cards

Table A.1 through Table A.6 show the class responsibility and collaboration (CRC) cards for this scenario.

Table A.1

CRC Card for Personalization

Personalization (Personalization) (EBT)		
Responsibility	**Collaboration**	
	Client	**Server**
Represents tailored information to the end user	• AnyParty • AnyMechanism	• represent Preference() • providePersonalizedView() • characterizeInformation()
Attributes: Content, site		

Table A.2

CRC Card for AnyParty

AnyParty(AnyParty) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
Any person or organization with some specific preferences	• AnyEntity • Personalization	• expressPreference() • selectChoices() • chooseOption()
Attributes: Name, contactInformation		

Table A.3

CRC Card for AnyCriterion

AnyCriterion (AnyCriterion) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
Provides the prerequisites for personalizing an entity	• AnyParty • AnyMechanism	• specifyRequirement() • stateCondition()
Attributes: criteria, agreement		

Table A.4

CRC Card for AnyMechanism

AnyMechanism (AnyMechanism) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
Various techniques employed for personalizing entities	• AnyCriterion • AnyView	• execute() • operate() • build()
Attributes: Linkage, assemble		

Table A.5

CRC Card for AnyView

AnyView(AnyView) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
It represents a way of showing the personalized results	• AnyCriterion • AnyMechanism	• scrutinize() • inspectOutlook() • illustrate()
Attributes: Size, viewLink, style		

Table A.6

CRC Card for AnyEntity

AnyEntity (AnyEntity) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
It represents an item or an entity which is being personalized	• AnyMechanism • AnyView	• create() • modify() • destroy()
Attributes: Type, name, characteristics		

A.3. Consequences

The Personalization analysis pattern is generic enough to serve as a building block for applications in diverse domains.

Using the Personalization analysis pattern to personalize AnyEntity requires AnyParty to provide valid preferences in terms of AnyCriterion. Moreover, the preferences of AnyParty must be made available before any request for personalizing AnyEntity. This does not mean that the pattern is incomplete. In fact, this is the nature of patterns to be dependent on other patterns or components.

A point of advantage with the Personalization analysis pattern is that the pattern has been derived based on the notion of software stability and will stand the test of time. A known drawback with this pattern is that it might result in incorrect or inaccurate results when personalization is not done properly. In addition, the privacy of AnyParty might be invaded when AnyParty attempts to define AnyCriterion.

The personalization pattern has the following benefits:

- **Flexibility**: An advantage of the Personalization analysis pattern is that it is flexible to accept all kinds of preferences from AnyParty. In case the supplied preferences are changed, the final product can be easily altered to reflect the new preferences.
- **Reusability:** The personalization analysis pattern is a stable pattern. It can be reused in many different scenarios across many different domains while the pattern remains unchanged.

A.4. Applicability with Illustrated Examples

In Section A.4, two examples to illustrate the use of Personalization analysis pattern are presented using some Unified Modeling Language (UML) use cases and a UML sequence diagram.

Application Number 1: Obtaining Personalized Results Using a Search Engine

In the context of searching the web using a search engine like Google's search engine, tuning the search results according to web user's preferences is called personalization. Considering a series of factors such as user's web navigation history, collected bookmarks, community behaviors, and certain web feedback components, this application could generate the results that are specific to what the web user meant to search for. Since the mechanism that personalizes the search results might require accessing and using personal data, the concept of information privacy is a key concern in implementation.

This application allows the visitors of a website to personalize the pages they view with various local news reports, local weather reports, and other information that users might be interested in. The mechanisms used to personalize the pages could include:

- **User profiling:** Using this mechanism, the data collected from a user's past activities and visited sites could be used to create a personalized web page.
- **Data analysis:** Various mechanisms could be used to intelligently access and analyze a user's prior activities to determine possible user's interests.

Model

The model for this application is shown in Figure A.2.

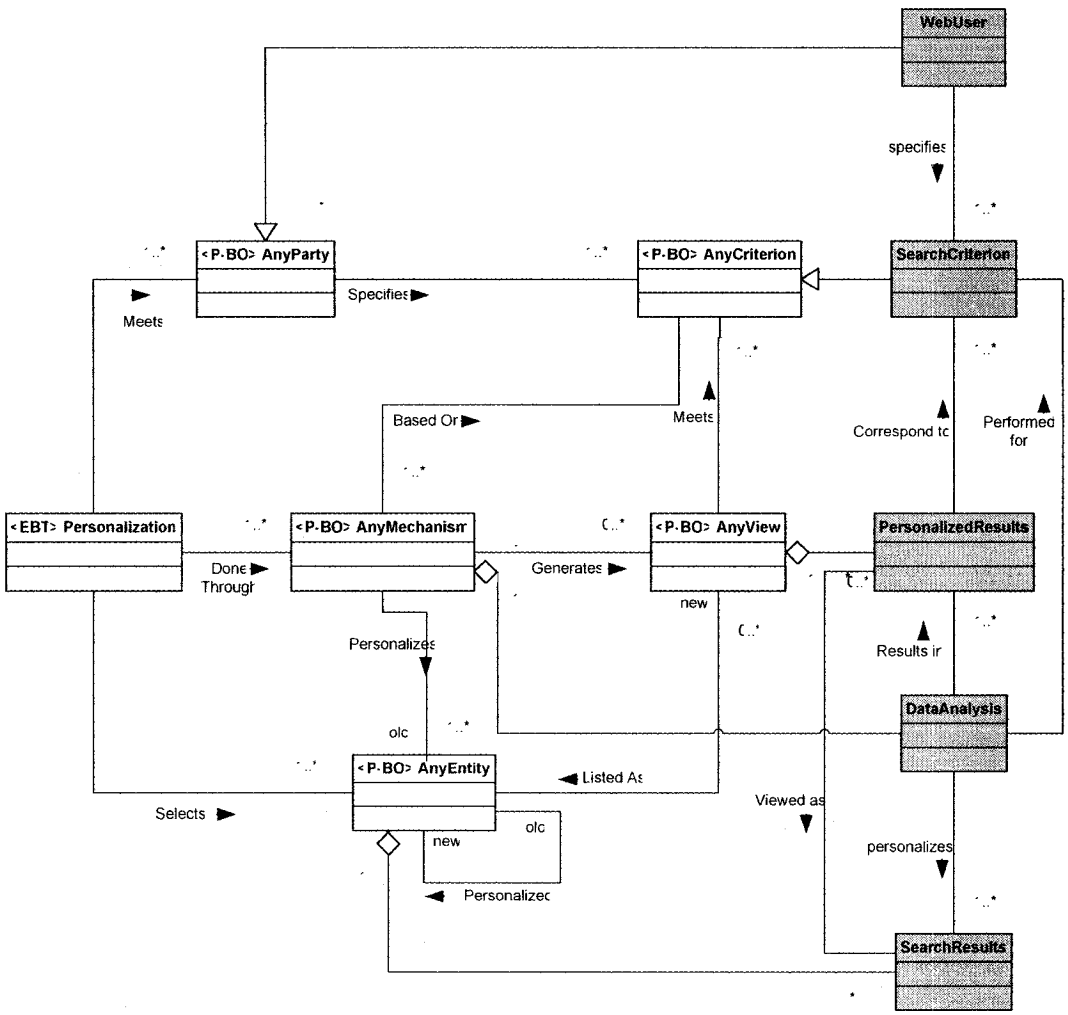

Figure A.2. Class diagram for personalized results by Google search engine.

Use Case

Use Case ID: 1.0

Use Case Title: Obtain Personalized Results Using a Search Engine

Table A.7 and Table A.8 show various components involved in web personalization scenario.

Table A.7

Actors and Roles for Personalization of Website

Actors	Roles
AnyParty	WebUser

Table A.8

Class Components for Web Personalization Use Case

Classes	Type	Attributes	Operations
Personalization	EBT	1. content • site	• providePersonalizedView()
AnyEntity	BO	• cause • effect • type	• characterizePrototype()
AnyParty	BO	• name • role • identity	• chooseOption() • selectChoices()
AnyMechanism	BO	• type • attribute • context	• generateView()
AnyCriterion	BO	• quality • value description	• specifyCharacteristic ()
AnyView	BO	• viewDescription • size • viewLink • style	• illustrate()
WebUser	IO	• interest • areaOfWork	• statePreference () • getPersonalizedResult()
SearchCriterion	IO	• hasPreference	• query() • searchParameter()
SearchResult	IO	• techniquesAvailable • displayType	• respondToSearchCriteria()
UserProfiling	IO	• noOfOutput • speed	• mechanosmForResults ()
DataAnalysis	IO	• model • type • make	• mechanosmForResults ()
PersonalizedResult	IO	• characteristic	• showPersonalizedPage()

Use Case Description

1. A WebUser (AnyParty) searches the web using Google search engine by providing SearchCriterion (AnyCriterion).

2. Personalization (EBT) is applied over SearchResult (AnyEntity).

3. DataAnalysis (AnyMechanism) and UserProfiling (AnyMechanism) use SearchResult (AnyEntity) to create PersonalizedResult (AnyView).

4. The PersonalizedResult (AnyView) is then made visible to WebUser (AnyParty) according to WebUser's (AnyParty) preferences.

5. Finally Personalization (EBT) provides the SearchResult (AnyEntity) to the WebUser (AnyParty).

Alternatives

- The WebUser (AnyParty) does not receive any personalized result.

Sequence Diagram

Figure A.3 shows the sequence diagram for the personalization scenario.

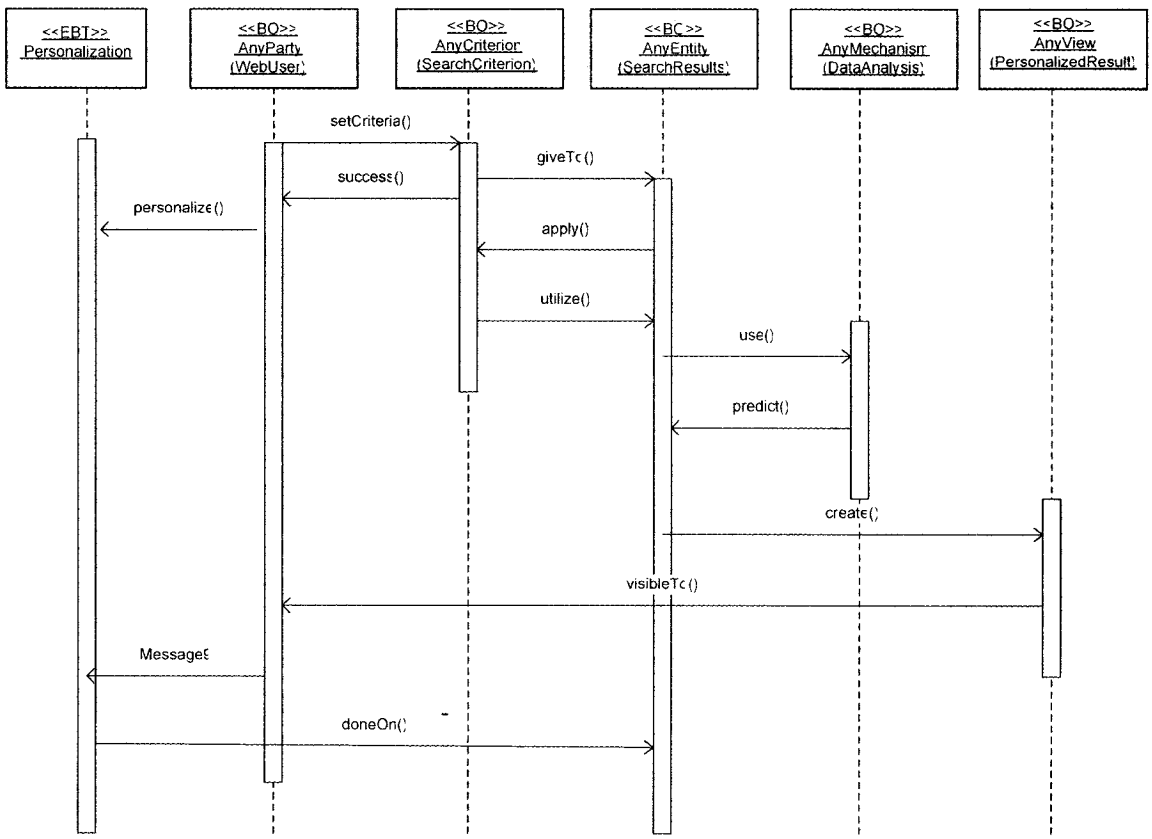

Figure A.3. Sequence diagram for personalized results from a search engine.

Sequence Diagram Description

The sequence diagram in Figure A.3 represents the flow of messages for personalizing the results of a web search. First, a WebUser (AnyParty) supplies a SearchCriterion (AnyCriterion) to the system. Then, the SearchCriterion (AnyCriterion) is utilized to produce SearchResult (AnyEntity). In turn, DataAnalysis (AnyMechanism) is used to create PersonalizedResult (AnyView) from the SearchResult (AnyEntity).

Finally, Personalization (EBT) provides the SearchResult (AnyEntity) to the WebUser (AnyParty), and the PersonalizedResult (AnyView) is made visible to the WebUser (AnyParty).

Application Number 2: Making a Personalized Sandwich at a Sandwich Store

Some sandwich stores provide their customer with the option to have their sandwiches personalized. In such stores, there usually exists a standard menu with different types of sandwiches. The following are possible formulas for making sandwiches, salads and wraps served at a sandwich store like SUBWAY® Sandwich store:

1.The customer can alter this formula or recipe, by choosing different vegetables, condiments, and breads.
2.The available vegetables include iceberg lettuce, tomatoes, red onions, green peppers, olives, and pickles.
3.The customer chooses the ingredients to be used in making the sandwich based on his or her personal preferences.
4.The sandwich store creates a sandwich for the customer based on the customer's preferences.

Model

The model for this application is shown in Figure A.4.

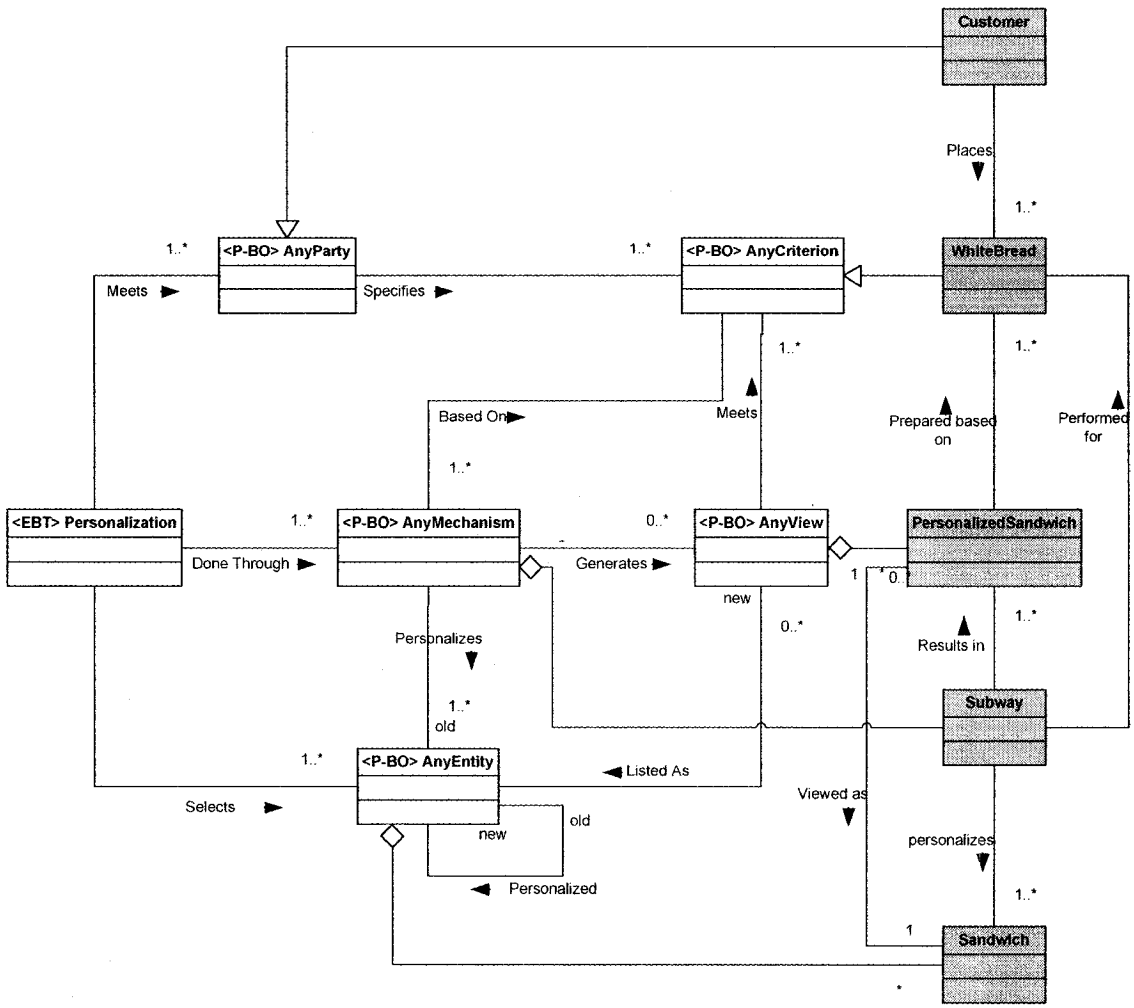

Figure A.4. Class diagram for creating a personalized sandwich.

Use Case

Use Case ID: 2.0

Use Case Title: Personalizing a Sandwich

Table A.9 and Table A.10 show different design components involved in sandwich personalization scenario.

Table A.9

Actors and Roles for Personalizing a Sandwich Use Case

Actors	Roles
AnyParty	Customer

Table A.10

Class Components for Personalizing a Sandwich Use Case

Classes	Type	Attributes	Operations
Personalization	EBT	2. content • site	• providePersonalizedView()
AnyEntity	BO	• cause • effect • type	• characterizePrototype()
AnyParty	BO	• name • role • identity	• chooseOption() • selectChoices()
AnyMechanism	BO	• type • attribute • context	• generateView()
AnyCriterion	BO	• quality • value description	• specifyCharacteristic()
AnyView	BO	• viewDescription • size • viewLink • style	• illustrate()
Customer	IO	• taste • preference	• statePreference() • getSandwichToView()
WhiteBread	IO	• type • size • amountOfCalorie	• option()
Condiment	IO	• type • amount • availability • amountOfCalories	• choice()
PersonalizedSandwich	IO	• satisfyCustomer • fatContent • calorie	• mechanismForSandwich ()
Subway	IO	• profit • noOfEmployees	• createSandwich()

Use Case Description

1. The Customer (AnyParty) goes to a Subway (AnyMechanism).

2. Customer (AnyParty) checks out the different options for AnyEntity (Sandwich) in the menu.

3. The Customer (AnyParty) orders AnyEntity (Sandwich).

4. The Customer chooses from a variety of breads, cheeses, fillings and sauces. One combination of preference by the Customer (AnyParty) might be white bread, mayonnaise, and veggie patty.

5. The Subway (AnyMechanism) makes the Sandwich (AnyEntity) based on the preferences stated by the Customer (AnyParty).

6. The PersonalizedSandwich (AnyView) is made available to the Customer (AnyParty).

Alternatives:

1. The Customer (AnyParty) does not receive the PersonalizedSandwich (AnyView).

Sequence Diagram

The sequence diagram for personalizing a sandwich is depicted in Figure A.5.

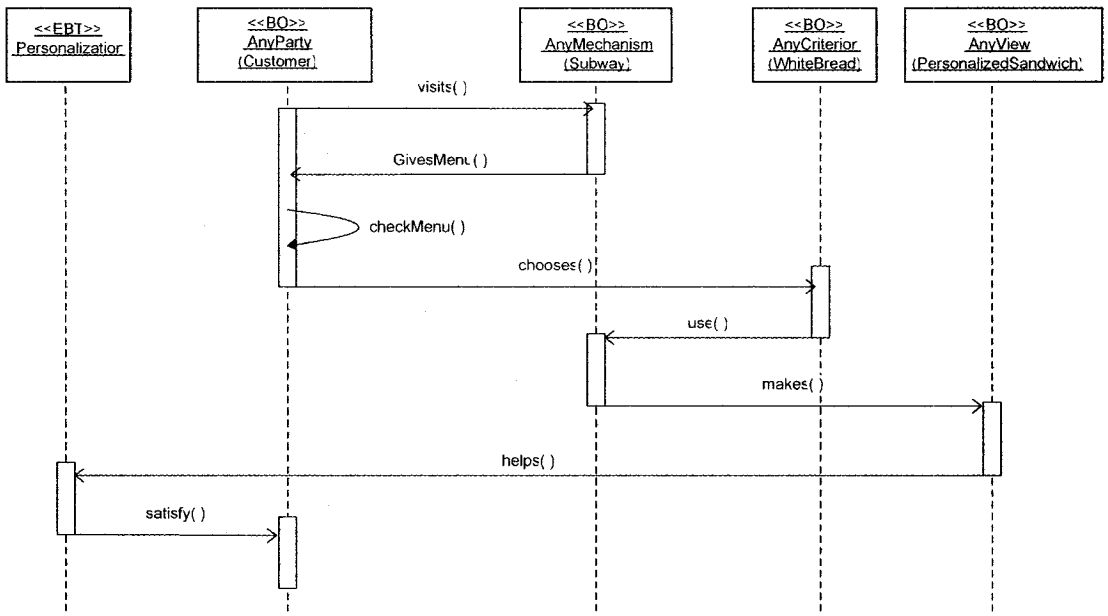

Figure A.5. Sequence diagram for personalizing a sandwich.

Sequence Diagram Description

The sequence diagram in Figure A.5 represents the flow of messages used to personalize a sandwich. In the scenario illustrated by Figure A.5, a Customer (AnyParty) goes to a Subway store in order to purchase a PersonalizedSandwich (AnyView). The Customer (AnyParty) uses the available menu and makes his or her own selections. The Customer (AnyParty) can choose from the options available like WhiteBread (AnyCriterion) and various Condiments (AnyCriterion). Then, the Subway (AnyMechanism) recipe is used to make a PersonalizedSandwich (AnyView). Hence, Personalization (EBT) is achieved, and the Customer (AnyParty) is satisfied with receiving the requested PersonalizedSandwich (AnyView).

A.5. Design and Implementation Issues

The Personalization analysis pattern is designed in such a way that it can be adapted and reused in diverse domains. Section A.5 describes some of the issues faced in designing and implementing the stable Personalization analysis pattern.

Developing a generic pattern to support multiple domains is not an easy task. It involves spending ample time for research to find all relevant BOs for the EBT of interest. Moreover, it would be difficult to capture all the underlying issues relating to various patterns that develop around the Personalization analysis pattern.

The EBTs and BOs are abstract parts of a stable pattern that do not change during implementation stage. They are the core to the model and remain stable.

Implementing a stable pattern is a time consuming task because developers must try to maximize the EBTs and BOs role in design so that developing the relevant IOs would be an easier task for users. The Personalization class is an EBT and has associative relationships with AnyParty (BO) and AnyEntity (BO). AnyParty (BO) will inherit various parties involved in any application context during the implementation phase.

A.6. Conclusion

The Personalization analysis pattern that is proposed in the research is based on the principles of the software stability model. Two different scenarios that use the Personalization analysis pattern are presented in Appendix A. Each object in the

Personalization analysis pattern has its own roles independent of any application to which this pattern will be applied. In terms of the mechanisms used in an application, more than one mechanism could exist to derive personalization. Extra care should be taken while choosing the appropriate mechanisms.

A difficult task is modeling the Personalization analysis pattern is devising a good class diagram to describe the pattern. It is essential to make the description as clear and accurate as possible so that the diagram would be beneficial in drawing the relevant sequence diagrams. Then, the process of creating the sequence diagrams would become much simpler and more flexible as they are translations of the class diagram. This is a key issue for getting good models.

The Personalization analysis pattern is flexible, and it can be applied to many scenarios. The industrial objects can be hooked to the BOs. However, the correct identification of EBTs and BOs for the Personalization analysis pattern is the most challenging task and requires some prior experience with software stability. Once the EBTs and BOs are correctly identified, the next challenge is to determine the relationships among the EBT and BOs such that the Personalization pattern could hold true in any context of managing content. Once this is ascertained, depending on the application at hand, the IOs could be attached to BOs. Thus, using the Personalization pattern as basis, many applications could be built by just hooking the application-specific IOs to the pattern. This results in reduced cost of development, faster time to market, and

stability in solution. Hence, the Personalization analysis pattern is very useful and beneficial to developers as well as users.

Appendix B: The Dynamism Stable Analysis Pattern

B.1. Introduction

Dynamism is a general concept that has several meanings and manifestations under different application domains. Generally speaking, the Dynamism stable analysis pattern aims at analyzing the concept of dynamism. Since the Dynamism pattern is developed using the Software Stability Model (SSM), it would be simple to employ this pattern in a number of different applications. This is made possible by just hooking the unstable Industrial Objects (IO) to the existing stable Business Objects (BO) according to users' needs. Appendix B introduces the Dynamism analysis pattern and attempts to model the core knowledge around the concept of dynamism. Appendix B also uses two separate scenarios to present the details of the Dynamism analysis pattern.

Since the concept of dynamism is understood differently in different contexts, the traditional and conservative approaches to software design will not result is a stable and reusable model. However, using the software stability model as introduced in Chapter One, the concept of dynamism could be represented in any context by using a single stable model. The software stability model requires creation of knowledge map by identifying the underlying Enduring Business Themes (EBT) and BOs. Furthermore, by hooking some IOs, which are application-specific design modules, the model can be further applied to any application domain. The resulting Dynamism analysis pattern would be stable, reusable, extendable, and adaptable. Thus any number of applications can be built by using this common model. In order to design a stable pattern, the

Dynamism analysis pattern attempts to capture the core knowledge of the concept of dynamism that is common to many applications and scenarios.

An analysis pattern is a special software pattern that is not related to a language or implementation. It captures the essential knowledge of a concept in abstract terms so that it can be reused over and over in various domains. Reuse of patterns helps in reducing software design and development time and controlling the prohibitive developmental costs. The objective of this document is to design a stable model for the concept of dynamism: the Dynamism analysis pattern. The knowledge captured in the Dynamism analysis pattern could then serve as a building block for modeling different applications in various domains. The rest of this document further presents the Dynamism analysis pattern using various scenarios.

B.2. Dynamism Analysis Pattern Document

The pattern presented in Section B.2 utilizes the software stability model to describe the notion of dynamism. Figure B.1 is the class diagram for the Dynamism analysis pattern.

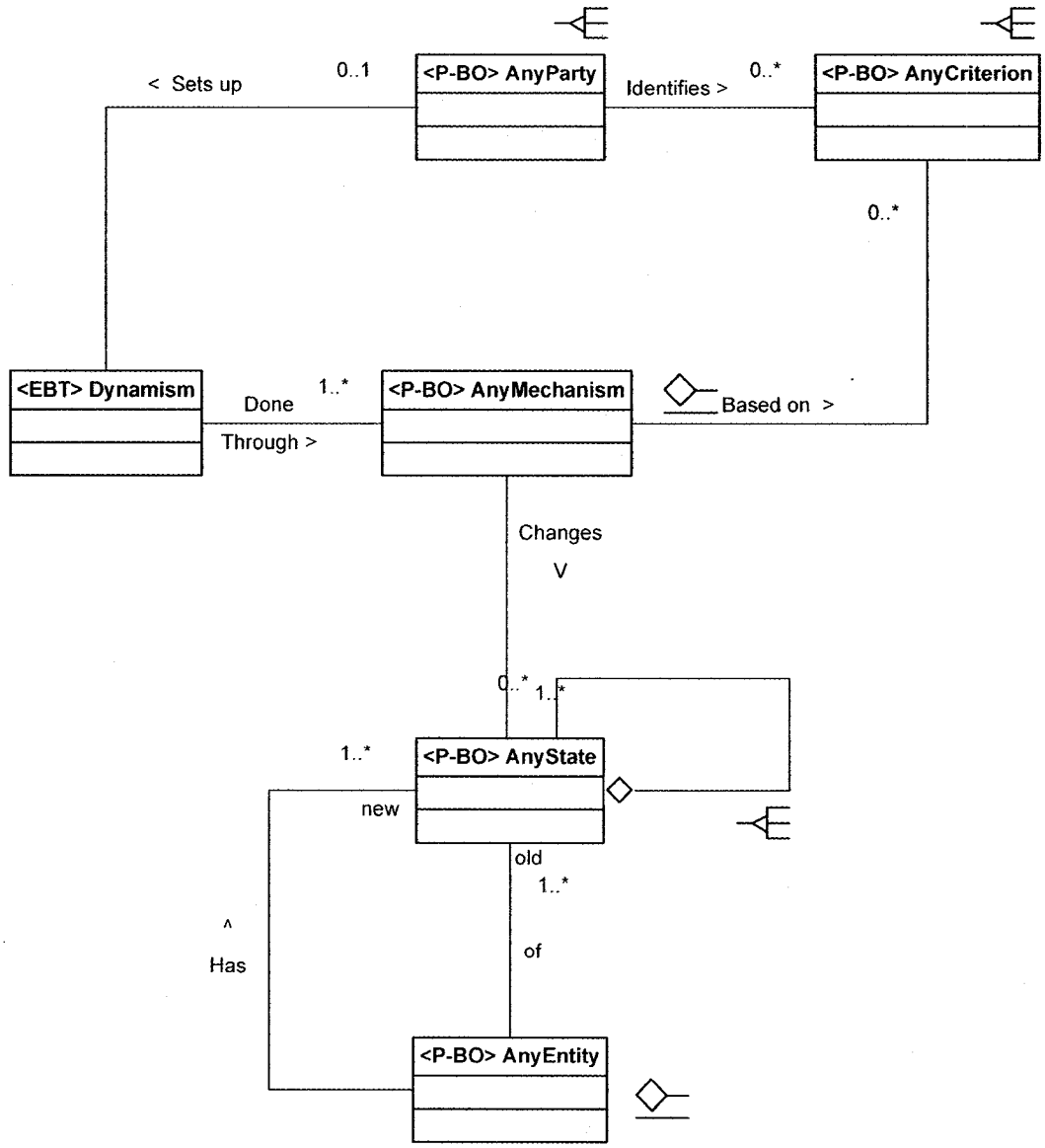

Figure B.1. Dynamism analysis pattern.

Pattern Structure and Participants

The participants of the Dynamism analysis pattern are as described below.

Classes:

Dynamism: Represents the notion of dynamism in a system. It represents the behavior and attributes which form and define dynamic changes in a system.

Patterns:

AnyEntity: This class represents the object on which dynamism is applied. Dynamism changes the sate of AnyEntity.

AnyParty: This class represents a person, organization, political party, country, or anyone who initiates the process of dynamism.

AnyMechanism: This class represents the mechanism through which the dynamic change happens to AnyEntity. In case of the dynamically updated websites, this class would be the coded module which updates the websites.

AnyState: This class represents the state of an entity in any point of time during the dynamism process.

AnyCriterion: This class represents the circumstances, prerequisite, and requirements under which AnyEntity is dynamically changed. Not all events may lead to a change to AnyEntity's state because only under the conditions changes will occur.

Class Diagram Description

A class diagram is a visual illustration of all classes in the model along with their static relationships with other classes. Description of the Dynamism analysis pattern's class diagram is presented below.

• Dynamism is the EBT of this model, and it helps to change one or more AnyEntity (BO). It dynamically changes the states of AnyEntity.

• AnyEntity (BO) has one or more AnyState (BO).

• AnyMechanism (BO) acts on AnyEntity (BO).

• AnyParty initiates the process of dynamism and further defines the criteria that influence how AnyEntity could dynamically change.

• AnyCriterion must be satisfied by AnyMechanism for Dynamism to follow.

CRC Cards

Tables B.1 through B.6 represent the class responsibility and collaboration (CRC) cards for the Dynamism analysis pattern.

Table B.1

CRC Card for Dynamism

Dynamism (Dynamism) (EBT)		
Responsibility	**Collaboration**	
	Client	**Server**
A process responsible for the change or motion of an entity or system	• AnyParty • AnyMechanism	• motion() • change() • process()
Attributes: process		

Table B.2

CRC Card for AnyEntity

AnyEntity (AnyEntity) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
It represents the entity that dynamically changes	• AnyState	• create() • modify() • destroy()
Attributes: Type, name, characteristic		

Table B.3

CRC Card for AnyParty

AnyParty (AnyParty) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
It represents a person or a group who initiates the process of dynamism	• Dynamism • AnyCriterion	• initiate() • identify() • monitor()
Attributes: Name, id, identity, phoneNumber		

Table B.4

CRC Card for AnyMechanism

AnyMechanism (AnyMechanism) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
The agent or means by which an effect or transition is produced	• Dynamism • AnyCriterion • AnyState	• provideProcedure() • promoteDynamism() • implementChange()
Attributes: purpose, privilege, step		

Table B.5

CRC Card for AnyCriterion

AnyCriterion (AnyCriterion) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
Provides the prerequisites and conditions for any entity undergoing dynamic changes	• AnyParty • AnyMechanism	• specifyRequirement() • statePreCondition()
Attributes: criteria, agreement		

Table B.6

CRC Card for AnyState

AnyState (AnyState) (BO)		
Responsibility	**Collaboration**	
	Client	**Server**
It represents the discrete individual stages of change in AnyEntity	• AnyEntity • AnyMechanism	• describeBehavior() • store() • traceTransformation()
Attributes: intialState, finalState		

B.3. Consequences

The Dynamism analysis pattern is generic enough to serve as a building block for applications in diverse domains.

Using Dynamism analysis pattern for any application will require AnyParty to supply correct conditions for an entity to undergo any change. Moreover, AnyParty involved in the process of dynamism must initiate the process of dynamism. The

Dynamism analysis pattern is built based on the software stability model's principles and offers the following advantages:

• The Dynamism pattern maintains a high level of abstraction and hence can be applied to many domains. The analysis pattern captures and encapsulates the concept of dynamism.

• AnyMechanism allows different mechanisms and ways to be implemented and used. However, an important drawback is that AnyMechanism might bring a software system to an invalid state when conditions are not specified accurately and clearly by users.

B.4. Applicability with Illustrated Examples

In Section B.4, two different scenarios are presented to clarify the underlying concept of dynamism in the Dynamism stable analysis pattern. The first scenario is about dynamism in a team as members of the team work together to complete a project. The second scenario is about dynamically changing web pages. These scenarios will try to flush out the intricate details connected to the Dynamism analysis pattern alone.

Application Number 1: Dynamics of a Team

A group of people join in a team to work on a project. This scenario shows how the Dynamism analysis pattern could be used to simulate the dynamically changing team. Table B.7 describes the concepts involved in the Dynamism analysis pattern.

Table B.7

Concepts Involved in Design of Dynamism Pattern

Concept	**Description**
Dynamism (EBT)	It represents the phenomenon of change.
AnyParty	In this scenario, the team members who change the team dynamically represent AnyParty.
AnyMechanism	It represents the means by which entities are dynamically changed. In this scenario, it is the team project or the process of working in a group that represents AnyMechanism.
AnyEntity	It represents an entity that is dynamically changing. In this scenario, it is the team work.
AnyCriterion	The needs and requirements to change AnyEntity dynamically. In this scenario, it is the project's deadline.
AnyState	It represents various states of AnyEntity.

Model

The model for this application is shown in Figure B.2.

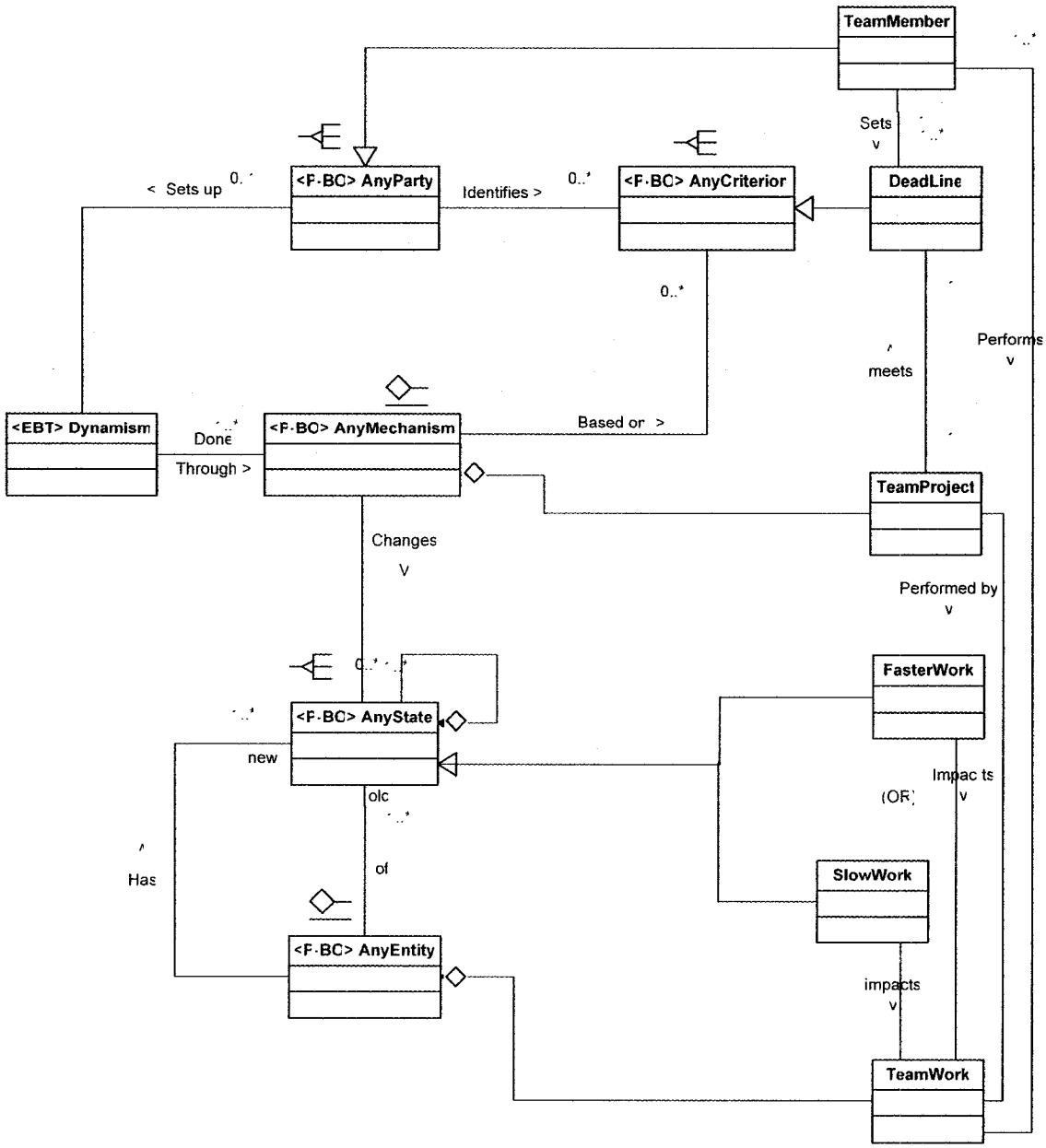

Figure B.2. Class diagram for dynamics of a team.

Use Case

Use Case ID: 1.0

Use Case Title: Dynamism in a Team

Table B.8 and Table B.9 show actors, roles, and class components for the Dynamism analysis pattern.

Table B.8

Actors and Roles for Dynamism in a Team Use Case

Actors	Roles
AnyParty	TeamMember

Table B.9

Class Components for Dynamism in a Team Use Case

Classes	Type	Attributes	Operations
Dynamism	EBT	• process	• start()
AnyParty	BO	• name • id • identity	• initiate() • setCriteria()
AnyMechanism	BO	• purpose • step	• changes()
AnyEntity	BO	• type • name • characteristic	• create() • update()
AnyCriterion	BO	• criteria • agreement	• statePreCondition() • informs()
AnyState	BO	• intialState • finalState	• traceTransformation() • update() • inform()
TeamMember	IO	• designation • department • responsibility	• perform() • interact()
Deadline	IO	• startDate • endDate • workAllocation	• provideDeadlineDetail()
TeamProject	IO	• id • name • noOfMembers	• satisfyCustomerNeed()
TeamWork	IO	• type • characteristic	• meetProjectDeadline() • alterDynamics()
SlowWork	IO	• name • dependencies • status • schedule	• currentStatus() • showProgress()
FasterWork	IO	• name • dependencies • status • schedule	• currentStatus() • showProgress()

Use Case Description

1. A TeamMember (AnyParty) works along other members in a team to complete a TeamProject (AnyMechanism).

2. The TeamMember (AnyParty) performs the TeamWork (AnyEntity) to complete the TeamProject (AnyMechanism). TeamMember (AnyParty) achieves TeamWork (AnyEntity) through working together on a project. When the Deadline (AnyCriterion) for the project nears, a TeamMember (AnyParty) works faster. The ongoing TeamWork (AnyEntity) moves smoothly and turns into a FasterWork (AnyState). A project Deadline (AnyCriterion) is specified by a member of a team or a project leader who is AnyParty. The TeamProject (AnyMechanism) would oversee and control the existing dynamism in the teamwork.

3. Due to dissatisfaction or some other external or internal forces, the members suddenly start quarreling, and the pace of the work slows down resulting in a SlowWork (AnyState).

4. The Dynamism (EBT) of the teamwork changes as specified by the TeamProject (AnyMechanism).

Alternatives

1. TeamMember (AnyParty) does not interact with other team members.

2. Deadline (AnyCriterion) cannot be met.

Sequence Diagram

Figure B.3 is the sequence diagram for this scenario.

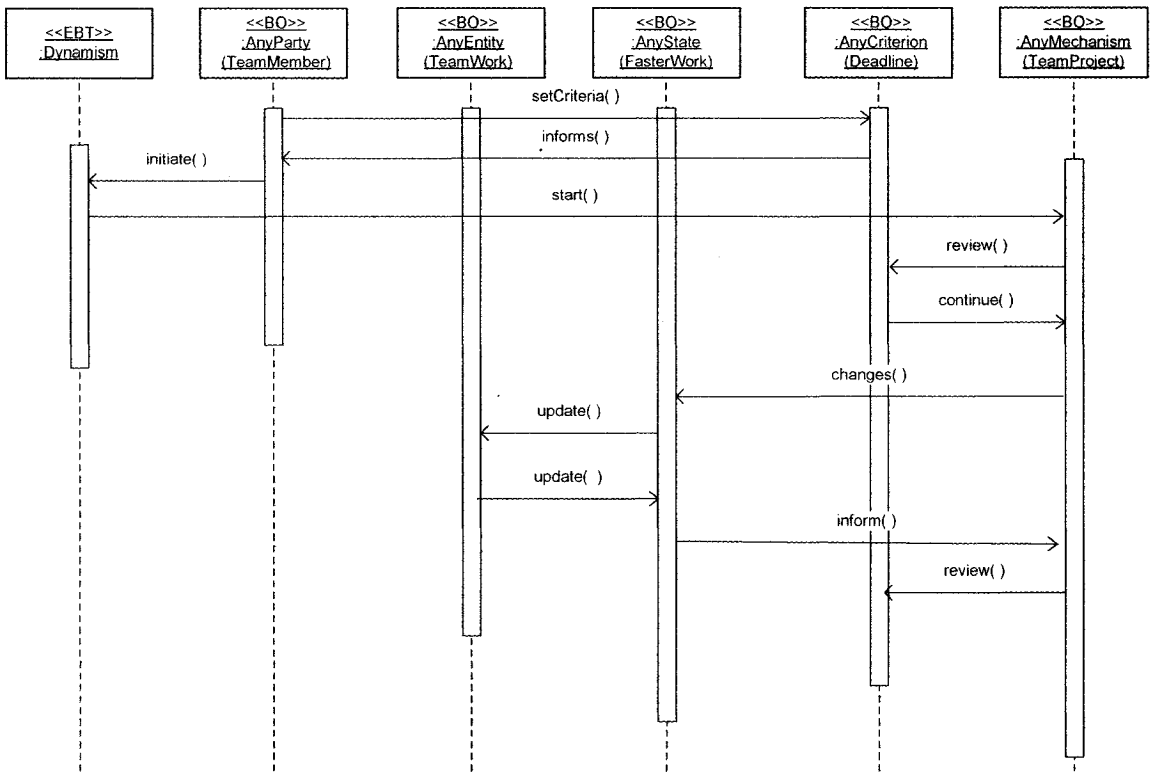

Figure B.3. Sequence diagram for dynamics of a team.

Sequence Diagram Description

Figure B.3 depicts the interactions of various classes. In this sequence diagram, TeamMember (AnyParty) defines and attempts to set Deadline (AnyCriterion) as the first step in the program execution. Then, Deadline (AnyCriterion) informs the TeamMember (AnyParty) about the success or conflicts in setting the new criterion by using the

informs() method. Then, after successfully setting the criterion, TeamMember (AnyParty) initiates the process of Dynamism. Dynamism in turn, invokes TeamProject (AnyMechanism) to proceed. The first thing TeamProject (AnyMechanism) would do is to review the Deadline (AnyCriterion).

If all criteria were met, the Deadline (AnyCriterion) will tell the system to proceed through the *continue()* method. Next, the TeamProject (AnyMechanism) would change AnyState of TeamWork (AnyEntity). Changing the state will be reflected on the TeamWork (AnyEntity), and the state of the TeamWork (AnyEntity) will be updated depending on the internal constructs and the existing triggers of AnyEntity. Then, AnyState will notify TeamProject (AnyMechanism) about the successful change of state of the TeamWork (AnyEntity). In the next step, the TeamProject (AnyMechanism) would review the Deadline (AnyCriterion) again for the next set of changes to be applied to the TeamWork (AnyEntity).

Application Number 2: Introducing Dynamism to Web

A common scenario in the web development domain is a unique scenario in which users desire to make their pages on the web look highly dynamic by periodically changing the background color, updating various content components, and showing certain contents based on some rules.

For simplicity, the following documented scenario is limited to dynamically changing a page's background color. In this scenario, a user accesses a website and attempts to set certain criteria to dynamically change a web page's background color

every ten minutes. In this scenario, dynamism is the phenomenon of change in background color of the page. The web user represents AnyParty, and the code module that enables the dynamic change in background color represents AnyMechanism. The web page certainly corresponds to AnyEntity. Finally, states of the page in terms of the change in color and links represent AnyState in this scenario. Table B.10 shows the concepts involved in the Dynamism analysis pattern.

Table B.10

Concepts Involved in Dynamism Applied to a Web Application

Concept	**Description**
Dynamism (EBT)	It represents the phenomenon of change.
AnyParty	In this scenario, the web users, who want to see their page dynamically change its background color, represent AnyParty.
AnyMechanism	It represents the means by which entities are dynamically changed. In this scenario, a web server changes the background color of web pages.
AnyEntity	It represents an entity that is dynamically changing. In this scenario, it is a page on the web.
AnyCriterion	The needs and requirements to change AnyEntity dynamically. In this scenario, the refresh time and the selected colors for background are encapsulated in a class called ColorSettings that represents AnyCriterion.
AnyState	AnyState in this scenario is the state of the page having a certain background color. In this scenario, the background color could be white or blue. The two colors are represented by WhiteBack and BlueBack classes.

Model

The model for this application is shown in Figure B.4.

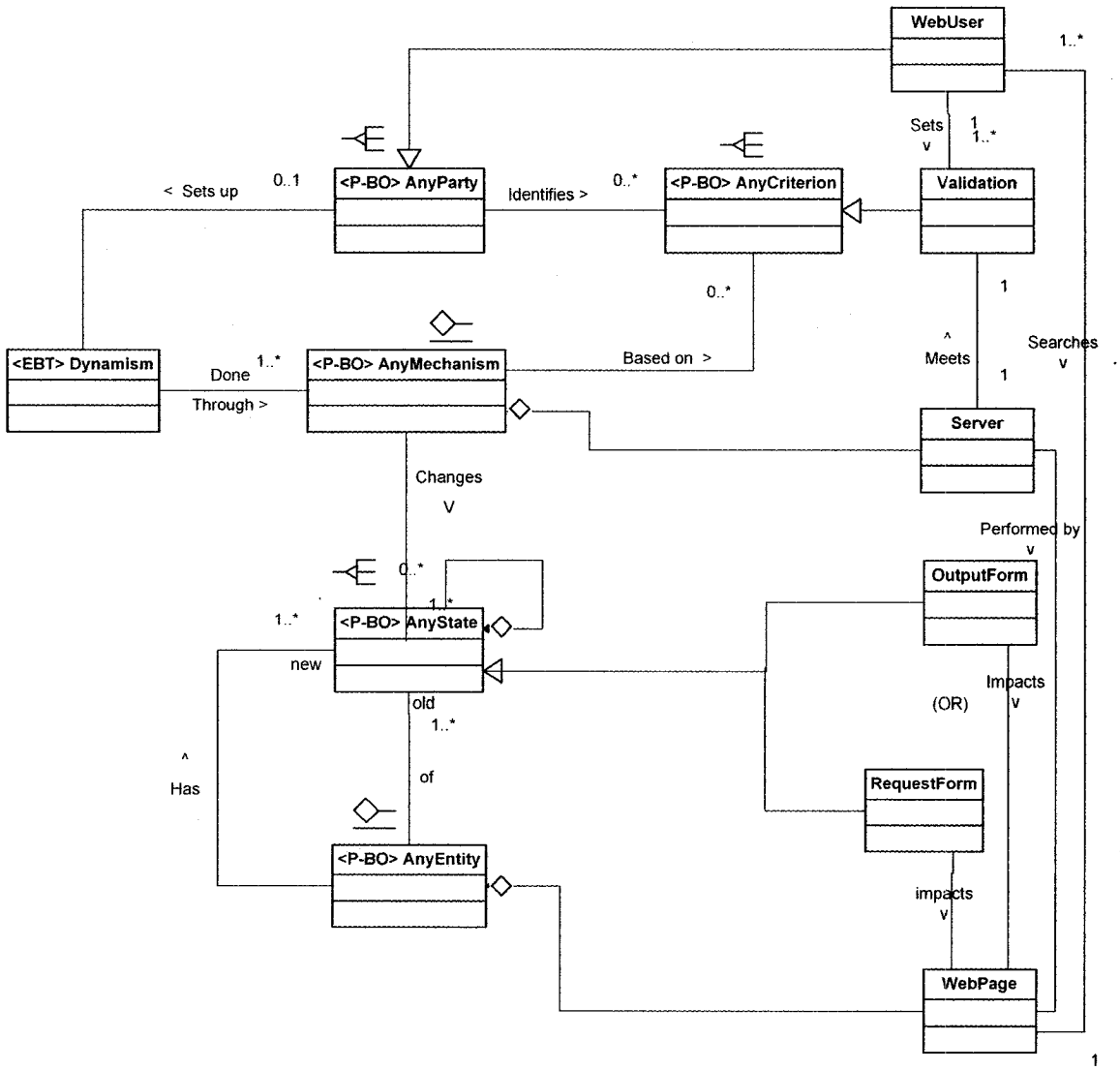

Figure B.4. Class diagram for creating a dynamic web page.

Use Case

Use Case ID: 2.0

Use Case Title: Create a Dynamic Web Page

Table B.11 shows all actors and their roles in this pattern while Table B.12 describes various classes involved in the Dynamism analysis pattern. As indicated in Table B.11, AnyParty is the sole actor of this pattern. However, for an application, which uses the Dynamism analysis pattern, a class such as AnyParty could be hooked to many IOs. In this scenario, a WebUser is an IO that could be hooked to AnyParty. Similarly, an instance of AnyEntity could be hooked to many IOs that represent some specific physical objects.

Table B.11

Actors and Roles for Web Usage Use Case

Actors	Roles
AnyParty	WebUser

Table B.12

Class Components for Web Usage Use Case

Classes	Type	Attributes	Operations
Dynamism	EBT	• process	• change()
AnyParty	BO	• name • identity • phoneNumber	• initiate()
AnyMechanism	BO	• purpose • privilege • step	• process () • checkCriterion() • reportResult()
AnyEntity	BO	• type • name • characteristic	• create() • updateState()
AnyCriterion	BO	• criterion • agreement	• setCriterion() • reportCriterion()
AnyState	BO	• intialState • finalState	• changeState() • showState() • report() • updateEntity()
WebUser	IO	• purpose • parameter	• addCriterion() • startDynamism()
ColorSettings	IO	• validationRule	• validate() • addColor() • removeColor() • currentColor()
ChangeModule	IO	• configuration	• actsOn() • processChange()
WebPage	IO	• type • use • information	• sendRequest() • refresh()
BlueBack	IO	• color • name	• takeUserInformation() • showInformation()
WhiteBack	IO	• color • name	• takeUserInformation() • showInformation()

Use Case Description

1. WebUser (AnyParty) chooses various colors and time intervals as ColorSettings (AnyCriterion). ColorSettings defines the background colors that the page would change into periodically. Color checking is done in this stage, and invalid colors will be rejected. The success will be reported to the WebUser (AnyParty).

2. When the WebUser (AnyParty) saves the ColorSettings (AnyCriterion), the command to start dynamism (Dynamism) is instantly issued.

3. As soon as Dynamism is initiated, ChangeModule (AnyMechanism) begins to dynamically change the background colors by querying for ColorSettings (AnyCriterion).

4. Next, the ChangeModule (AnyMechanism) would change the background color of the page after an specified time interval and changes the state of WebPage (AnyEntity) by switching the current state to WhiteBack (AnyState) or BlueBack (AnyState).

5. Finally, the ChangeModule (AnyMechanism) returns the thread of execution to Dynamism (EBT).

Alternatives

1. The WebUser (AnyParty) does not enter correct color criterion.

2. ChangeModule (AnyMechanism) is not able to switch between WhiteBack (AnyState) and BlueBack (AnyState).

Sequence Diagram

The sequence diagram for making a dynamic website is shown in Figure B.5.

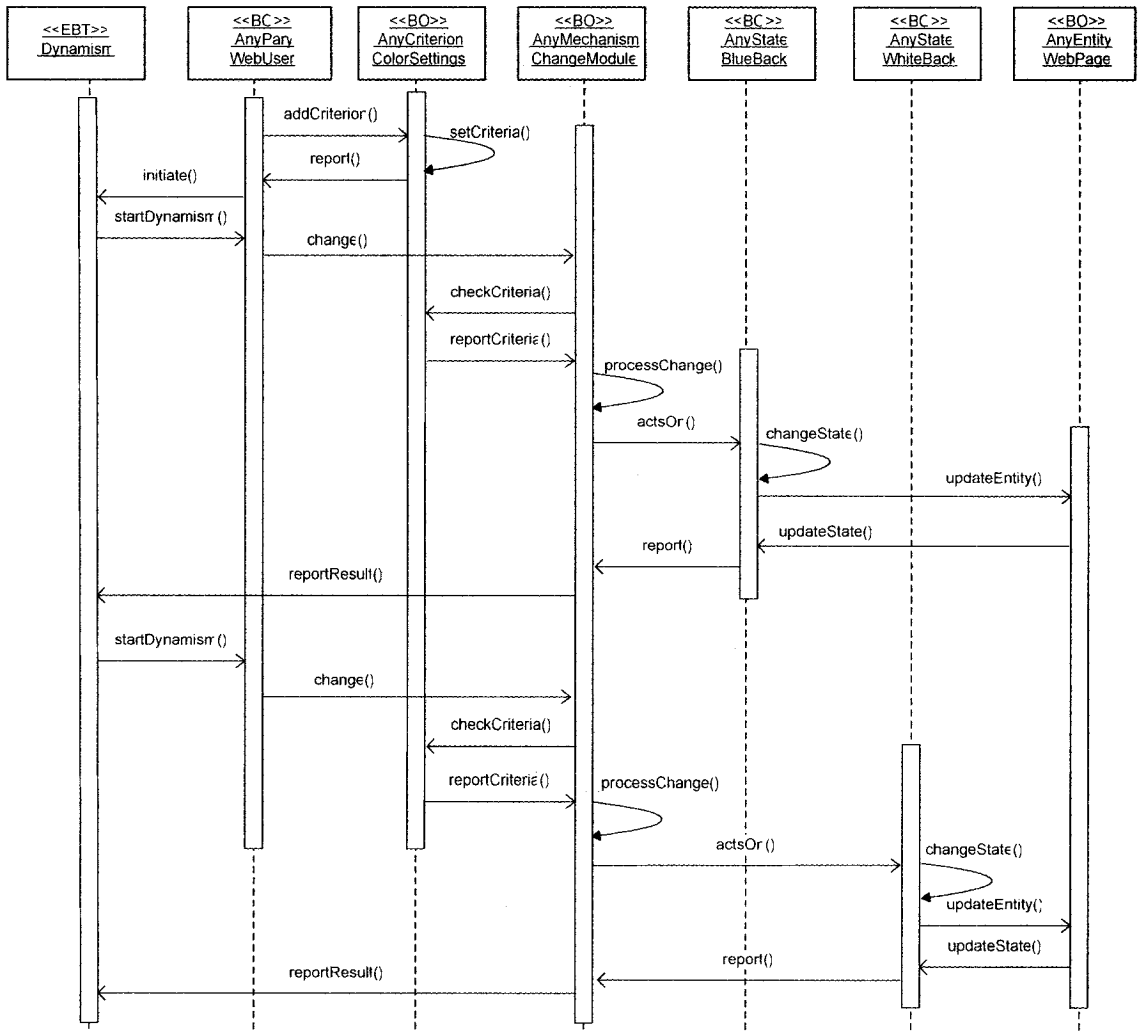

Figure B.5. Sequence diagram for dynamism in a web page.

Sequence Diagram Description

The sequence diagram in Figure B.5 represents the flow of messages when a WebUser (AnyParty) starts setting some criteria for making a web page dynamic. The WebUser (AnyParty) starts choosing the desired colors and a time interval to keep the current page color before updating the page again. As soon as the WebUser (AnyParty) defines ColorSettings (AnyCriterion), the WebUser (AnyParty) would be notified about the success or failure of the operation. On attaining success, the WebUser (AnyParty) initiates dynamism (Dynamism). In turn, Dynamism invokes ChangeModule (AnyMechanism) for the process of dynamism. One of the primary checks that ChangeModule (AnyMechanism) performs is checking against ColorSettings (AnyCriterion). In turn, AnyCriterion would report the criteria to ChangeModule. Then, ChangeModule (AnyMechanism) would change the state of the WebPage (AnyEntity) to BlueBack (AnyState). Then, the BlueBack (AnyState) would inform the WebPage (AnyEntity) about the change. Later, the WebPage (AnyEntity) will change its background color, and it notifies the AnyState that it is successfully in the BlueBack state. Next, the BlueBack (AnyState) informs the ChangeModule (AnyMechanism), and that would let Dynamism (EBT) know about the success. At this point, the thread of execution returns back to Dynamism (EBT). Using the same process, after ten minutes, the ChangeModule (AnyMechanism) will request for another update to page. This time, the state would be changed to WhiteBack, and again, the thread of execution will return to Dynamism (EBT).

B.5. Design and Implementation Issues

The Dynamism analysis pattern has been created based on the principles of software stability. This pattern could be used to model the concept of dynamism.

The software stability model applies the concept of stability onto the software design process through the use of EBTs and BOs. The Dynamism analysis pattern could be used for any application by hooking the relevant IOs to the pattern. The IOs are not reusable in different domains as they are application-specific.

All EBTs and BOs involved in a stable pattern are identified in the design phase of the stable software development process. The implementation phase in development process is heavily dependent on using the stable patterns, and the deployment phase would involve attaching the IOs to the implemented patterns. Since all EBTs and BOs are available in tandem while modeling a stable pattern, the concurrent software development approach could be employed in the development process. One of the advantages of using the software stability model is the reduction in software maintenance costs due to stability of the software and the reduced need for maintenance.

The Dynamism analysis pattern must be designed in a way that it could satisfy all the relevant requests made by AnyParty to dynamically change AnyEntity. Designing a generalized pattern that would be applicable to all relevant business domains is a time consuming process. This is due to the fact that designing a stable pattern requires identifying and recognizing the existing relations among all possible EBTs and BOs. In

addition, a stable pattern could be designed only through a comprehensive knowledge search and processing.

An important point in implementing all stable patterns is to ensure that there is no connection between the pattern EBTs and application IOs. Dynamism (EBT) has no relation with IOs. It has only associative relationships with AnyParty (BO) and AnyMechanism (BO). AnyParty (BO), as a class, will inherit various parties involved in the application context.

B.6. Conclusion

The unique concept of dynamism is modeled based on the principles of the software stability model. This model can be used in different domains as the application-specific components can be changed according to the application domain's requirements. The proposed model captures the core knowledge of dynamism as it is used in different applications. The captured knowledge in the form of a pattern is presented as a collection of EBTs and BOs.

Building a stable analysis pattern for the concept of dynamism is a difficult task and requires thorough understanding of the problem. The Dynamism pattern is modeled using the software stability model and it results in a reusable, extensible, and stable pattern.

The correct recognition of EBTs and BOs for a pattern is a challenging task, which demands considerable amount of time and expertise. Once the EBT and BOs are correctly identified, the next challenge would be determining the relationships among the

pattern's EBTs and BOs. Once this is done, some application IOs must be attached to the BOs. Thus, using the Dynamism pattern as a basis, many applications could be built by simply connecting all application-specific IOs to the pattern. This will ultimately result in reduced software costs, shorter development cycles, and a stable solution.

Appendix C: Sample Code

C.1. Introduction

In Appendix C, sample Java codes used for patterns implementation are listed. For implementation, Java Development Kit (JDK) of version 2.6 is used in Sun Microsystems' NetBeans 5.5.1 environment. This appendix lists the sample codes in four tables. As commonly used throughout this thesis report, EBT stands for Enduring Business Theme, BO stands for Business Object, and IO stands for Industrial Object.

C.2. Sample Code: AnyCriteria

The following sample code in Table C.1 belongs to AnyCriteria (BO). AnyCriteria is also known as AnyCriterion. AnyCriteria is used in both Dynamism and Personalization analysis patterns. Moreover, AnyCriteria is used in many other patterns. Discussing those patterns is beyond the realm of discussions of this report. Sections of the actual implementation code are omitted for simplicity.

Table C.1

Sample Code for AnyCriteria Business Object

```
package personalizationdemo;


import java.lang.*;
import java.util.*;
```

Table C.1 continued...

```
public class AnyCriteria
{
        //ArrayList<AnyMechanism> mechanismList = new ArrayList<AnyMechanism>();


        private String criteriaName;
        private String description;
        protected ArrayList requirements = new ArrayList();
        // set requirements like "10 items allowed in a room", "only 2 chairs allowed in a room" etc
        private Map mechanismsSupported;
        int id;
    String conditions;
    String property;
    int priority;
        public AnyCriteria()
        {
         id = -1; // this is default uninitialized id
         criteriaName = "undefined";
         priority = -1; //undefined priorities
                mechanismsSupported = new HashMap();
        }


        public AnyCriteria(int id_in, int priority_in, String name_in)
        {
         id = id_in;
```

Table C.1 continued...

```
        criteriaName = name_in;
        priority = priority_in;
                mechanismsSupported = new HashMap();
      };
public AnyCriteria coloneIt(){

   AnyCriteria new_cr = new AnyCriteria();
   new_cr.criteriaName = criteriaName;
   new_cr.conditions =conditions;
   new_cr.description = description;
   new_cr.id = id;
   new_cr.mechanismsSupported = mechanismsSupported;
   new_cr.priority = priority;
   new_cr.property = property;
   new_cr.requirements =requirements;
   return new_cr;
}
//sets a priority level for the criteria. This is relative to other criteria in the system,
public void prioritize(int new_priority){
   priority = new_priority;
   return;
};

    public String getName()
```

Table C.1 continued...

```
    {
            return criteriaName;
     };



public void setId (int id_in){
   id= id_in;
   return;
};

public void setName (String name_in){
   criteriaName= name_in;
   return;
};

public void setPriority (int pri_in){
   priority = pri_in;
   return;
};
// defines the criteria.
public void define(){return;};

// verifies the criteria to see if it is actually valid.
// this is done by comparing the user suuplied criteria through cr_in against
```

Table C.1 continued...

```
    // this class's criteria
    public int verify(AnyCriteria cr_in){

     //further implementation to be done as per need
       return 0;
    };
    //selects the criteria to be used.
    public void apply() {return;};

    //parses the conditions that are associated with the criteria
    public void parse(){return;};

    //show the bases of where the criteria came from.
    public void exhibit( ) {return;};

}
```

C.3. Sample Code: RoomCriteriaWinodw

The next sample code in Table C.2 corresponds to the GUI window software implementation that would collect the criteria predicates set by AnyParty in the Personalization analysis pattern implementation. Sections of code are omitted for simplicity. The entire GUI has been designed by the author.

Table C.2

Sample Code for a GUI Window to Receive RoomCriteria from User

```
package room;

import personalizationdemo.*;
import java.lang.*;
import java.util.*;




public class RoomCriteriaWindow extends javax.swing.JFrame {



   //Class members:
   public RoomCriteria roomcriteria;



   /**
    * Creates new form RoomCriteriaWindow
    */
   public RoomCriteriaWindow() {
      roomcriteria = new RoomCriteria();
      initComponents();
       //System.out.println(  "While doing it, room name is: " + roomcriteria.getName() );
   }
```

Table C.2 continued...

```
public RoomCriteriaWindow(int id_in, int priority_in, String name_in ) {
    roomcriteria = new RoomCriteria(id_in,priority_in,name_in );
    initComponents();
  }

  // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
  private void initComponents() {
    panel1 = new java.awt.Panel();
    jScrollPane2 = new javax.swing.JScrollPane();
    jPanel1 = new javax.swing.JPanel();
    jPanel2 = new javax.swing.JPanel();
    jLabel2 = new javax.swing.JLabel();
    jComboBox2 = new javax.swing.JComboBox();
    jButton2 = new javax.swing.JButton();

   //sample code omitted.....

    jScrollPane4 = new javax.swing.JScrollPane();
    jList2 = new javax.swing.JList();
    jLabel7 = new javax.swing.JLabel();
    jLabel8 = new javax.swing.JLabel();
    jLabel9 = new javax.swing.JLabel();
    jScrollPane3 = new javax.swing.JScrollPane();
    jEditorPane1 = new javax.swing.JEditorPane();
```

Table C.2 continued...

```
        jLabel3 = new javax.swing.JLabel();


        org.jdesktop.layout.GroupLayout panel1Layout = new org.jdesktop.layout.GroupLayout(panel1);
        panel1.setLayout(panel1Layout);
        panel1Layout.setHorizontalGroup(
            panel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 100, Short.MAX_VALUE)
        );
        panel1Layout.setVerticalGroup(
            panel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(0, 100, Short.MAX_VALUE)
        );
        jScrollPane2.setBackground(new java.awt.Color(255, 255, 255));
        jScrollPane2.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0),
2));
        jScrollPane2.setForeground(new java.awt.Color(0, 51, 102));


        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Room Criteria Definition Window");
        setAlwaysOnTop(true);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                formWindowClosing(evt);
            }
        });
```

Table C.2 continued...

```
jPanel2.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0), 2));

jLabel2.setText("Set Max Number of Items Allowed in Room:");


jComboBox2.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "0", "1", "2",
"3", "4", "5", "6", "7", "8", "9", "10", "11", "12" }));


jButton2.setText("Set Criteria");

jButton2.addActionListener(new java.awt.event.ActionListener() {

   public void actionPerformed(java.awt.event.ActionEvent evt) {

      jButton2ActionPerformed(evt);

   }

});

org.jdesktop.layout.GroupLayout jPanel2Layout = new

org.jdesktop.layout.GroupLayout(jPanel2);

jPanel2.setLayout(jPanel2Layout);

jPanel2Layout.setHorizontalGroup(

//sample code omitted....

        .add(jPanel2Layout.createSequentialGroup()

           .addContainerGap()

           .add(jLabel2))

        .add(jPanel2Layout.createSequentialGroup()

           .add(25, 25, 25)

           .add(jComboBox2, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)

           .add(99, 99, 99)
```

Table C.2 continued...

```
                .add(jButton2)))
            .addContainerGap(192, Short.MAX_VALUE))
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .add(jLabel2)
                .add(18, 18, 18)
                .add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jButton2)
                    .add(jComboBox2, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(83, Short.MAX_VALUE))
        );
        jPanel3.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0), 2));
        jLabel1.setText("Set Items Count Criteria:");

        jList1.setModel(new javax.swing.AbstractListModel() {
            String[] strings = { "Adj. Chairs", "Visitor Chair", "Main Desk", "Side Desk", "Study Light",
"Side Cabinet", "Desktop CPU", "LCD Monior", "Bookcase", "Visitor Table" };
            public int getSize() { return strings.length; }
            public Object getElementAt(int i) { return strings[i]; }
        });
        jScrollPane1.setViewportView(jList1);
```

Table C.2 continued...

```
jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "0", "1", "2", "3",
"4", "5", "6" }));


        jButton1.setText("Set Criteria");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });


// Omited code for Simplicity .....


        pack();
    }// </editor-fold>



    //personalizing wall color
    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
            String valueSelected = (String) jList2.getSelectedValue();
            roomcriteria.setWallColor(valueSelected);
            jEditorPane1.setText("Selected Room Wall Color: " + valueSelected );
    }



    //personalizing total allowed count of each item in the room
```

Table C.2 continued...

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String valueSelected = (String) jComboBox1.getSelectedItem();
        int i = Integer.parseInt(valueSelected.trim());
        String keySelected = (String) jList1.getSelectedValue();
        roomcriteria.addItemMax(valueSelected,i);
        jEditorPane1.setText("Selected Item: " + keySelected + " for the count of: " + valueSelected );
}
// this for seeting max number of items allowed in the room
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        String valueSelected = (String) jComboBox2.getSelectedItem();
        int i = Integer.parseInt(valueSelected.trim());
        jEditorPane1.setText("Selected Max Items: " + valueSelected );
        roomcriteria.setMaxAllowedItems(i);
}

  private void formWindowClosing(java.awt.event.WindowEvent evt) {
  this.setVisible(false);
}
/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
  java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
      new RoomCriteriaWindow().setVisible(true);
```

Table C.2 continued...

```
        }
    });
}
```

C.4. Sample Code: RoomCriteria

The next sample code corresponds to the RoomCriteria Industrial Object which should be connected to AnyCriteria BO as shown in Table C.3.

Table C.3

Sample RoomCriteria Code

```
package room;

import personalizationdemo.*;

import java.lang.*;

import java.util.*;


public class RoomCriteria extends AnyCriteria{


   int maxAllowedItems;                // max number of allowed items

   HashMap<String,Integer> itemsMap;    //items specified by user

   HashMap<String,Integer> refMapMax;   //max num of allowed items in room

   HashMap<String,Integer> refMapMin;   //min num of allowed items in room

   ArrayList wallColors;               //possible colors of the room

   String wallcolor;
```

Table C.3 continued...

```
/**
 * Creates a new instance of RoomCriteria
 */
public RoomCriteria() {
   super();
   itemsMap = new HashMap<String,Integer>();
   maxAllowedItems = 10;
   refMapMax = new HashMap<String,Integer>();
   refMapMin = new HashMap<String,Integer>();
   wallColors = new ArrayList();
};

public RoomCriteria(int id_in, int priority_in, String name_in ) {
        super(id_in, priority_in,name_in );
        itemsMap = new HashMap<String,Integer>();
        maxAllowedItems = 10;
        refMapMax = new HashMap<String,Integer>();
        refMapMin = new HashMap<String,Integer>();
        wallColors = new ArrayList();
};


public void setWallColor (String color){
  wallcolor = color;
};
```

Table C.3 continued...

```
public String getWallColor()
{
 return wallcolor;
}

public void setMaxAllowedItems (int s){
  maxAllowedItems = s;
};

public int getMaxAllowedItems()
{
 return maxAllowedItems;
}
 public void addItemMax (String item, int max){
  refMapMax.put(item, max);
};

public void addItemMin (String item, int min){
  refMapMin.put(item, min);
};

public void addItemColor (String item, String Color){
 return;
}
```

Table C.3 continued...

```
public int verifyItemLimits (String item, int count ){

    if ( ((Integer) (refMapMin.get(item) ) <=  (Integer)count ) &&  ( (Integer)count <=
(Integer)refMapMin.get(item))  ){

       return 0;

    }

    else return -1;

  };


  public int verifyTotalItems (int count){

    if (maxAllowedItems> count){

      return 1;

    }else {

      return -1;

    }

  };


  public int verifyColor (String item, String color){

    return 0;

  };

}
```

C.5. Sample Code: AnyMechanism

The sample code in Table C.4 belongs to AnyMechanism BO with sections of code omitted for simplicity.

Table C.4

Sample Code for AnyMechanism Business Object

```
package personalizationdemo;


import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;


public class AnyMechanism
{


        private String mechanismName;
        private String mechanismDescription;
        protected AnyEntity entityToBeModified;
        public HashMap<String,Object> mechanisms_criteria;
// map of all mechanisms and their associated criteria.
 //later, we could make this private but implement all needed methods
    public HashMap<String,Object> mechanism;


    String context;
    int id;
    int status;
    String application;
```

Table C.4 continued...

```
String components;
        String description;


          public AnyMechanism()
          {
        mechanismName ="undefined";
        mechanismDescription ="undefined";
        id = -1;
        status = 0;
        application ="undefined";
        components ="undefined";
        description ="undefined";
        mechanisms_criteria = new HashMap<String,Object>();
        mechanism = new HashMap<String,Object>();
          }
          public AnyMechanism(int id_in, int status_in, String name_in)
          {
        mechanismName = name_in;
        mechanismDescription ="undefined";
        id = id_in;
        status = status_in;
        application ="undefined";
        components ="undefined";
        description ="undefined";
        mechanisms_criteria = new HashMap<String,Object>();
```

Table C.4 continued...

```
    mechanism = new HashMap<String,Object>();


    }


public void addMechanism (String mech_name, Object mech){
  mechanism.put(mech_name, mech);
}


public Object getMechanism (String name){
  return mechanism.get(name);
}


public int executeWithCriteria( HashMap mech_cri, AnyEntity entity) {


  if (mech_cri.isEmpty() || mechanisms_criteria.isEmpty()){
    return -1;
  }
  // 1. check and match the user criteria gainst refrence criteria
  // user criteria is passed through mech_cri


  Iterator keySetIterator = mech_cri.keySet().iterator();
  while(keySetIterator.hasNext()) {
    Object key_in = keySetIterator.next();
    //System.out.println( key_in);
    // add additional type checking here or use java 5 templated form... a future task
```

Table C.4 continued...

```
            if (( (AnyCriteria) (mechanisms_criteria.get(key_in))).verify( ( (AnyCriteria)
mech_cri.get(key_in))) != 0) {

              return -1;

            }

          }


          // 2. now that criteria are matched, apply the mechanism

          // it is expected the execute function to be over-written written by IOs

          return execute(entity);

        };


        public int execute(AnyEntity entity) {

          //now apply each mechanism on entity here

          return 0;

        };
// sections of code omitted here.....

  }
```

Appendix D: Demonstration

D.1. Introduction

In Appendix D, the Personalization analysis pattern, which is designed and discussed in this thesis report, is demonstrated as part of a larger demonstration of the system of patterns that constitute the Unified Content Management Engine (UCME). For implementing this demonstration, Java version 2.6 is used in Sun Microsystems' NetBeans 5.5.1 environment under Windows Vista Ultimate 64-bit operating system and over 64-bit Intel Centrino Duo platform.

The scenario demonstrated in Appendix D is about personalizing a room by changing the wall color, changing the room name, and taking items in and out of the room. The room is an Industrial Object (IO) connected to AnyEntity Business Object (BO). Other components involved in this demonstration are listed in Table D.1.

Table D.1

List of Industrial Objects for the Personalization Pattern Components

Pattern Component	**Industrial Object**
AnyEntity(BO)	RoomEntity
AnyCriteria(BO)	RoomCriteria
AnyParty(BO)	User
AnyMechanism(BO)	RoomMechanism
AnyView(BO)	RoomView

In Appendix D, the focus is on demonstration of the implemented software using software stability model and based on the designs proposed in this thesis report. The implementation techniques, therefore, are not discussed.

D.2. Demonstration

The implemented application is first launched to personalize a room as AnyEntity. Upon the application invocation, the main window for personalization is launched. This window is a dashboard that would allow the user as AnyParty to set the RoomCriteria as AnyCriteria, view the RoomView and personalize the room. Figure D.1 shows the main window or the personalization dashboard.

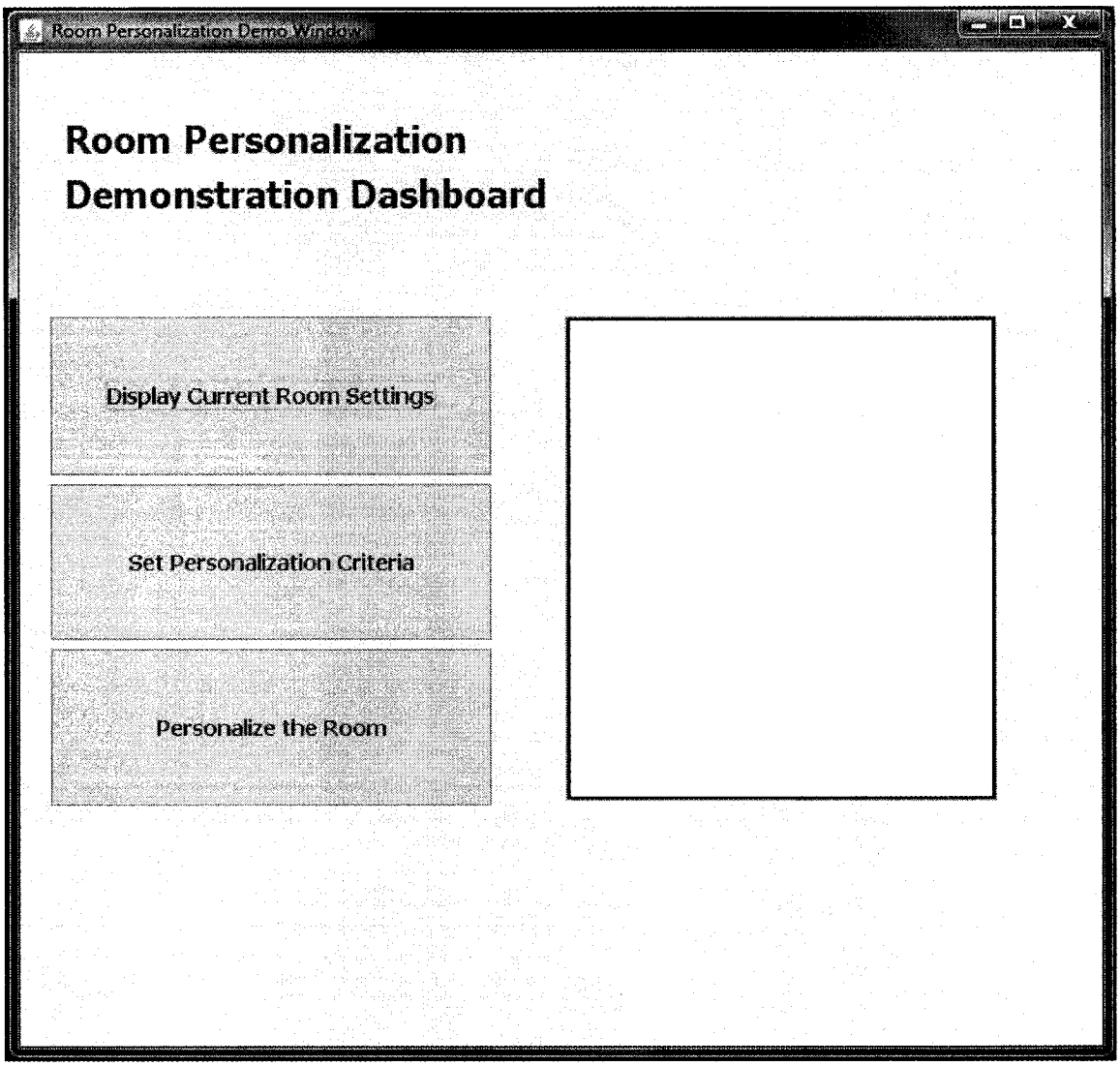

Figure D.1. The room personalization demonstration dashboard.

The next step is about invoking a new window for viewing the room. In this demo, the empty room is displayed first by clicking on the top button in the main window. This would launch the next window to show the RoomView. The implemented view is a textual view of the room. Figure D.2 shows a non-personalized empty room.

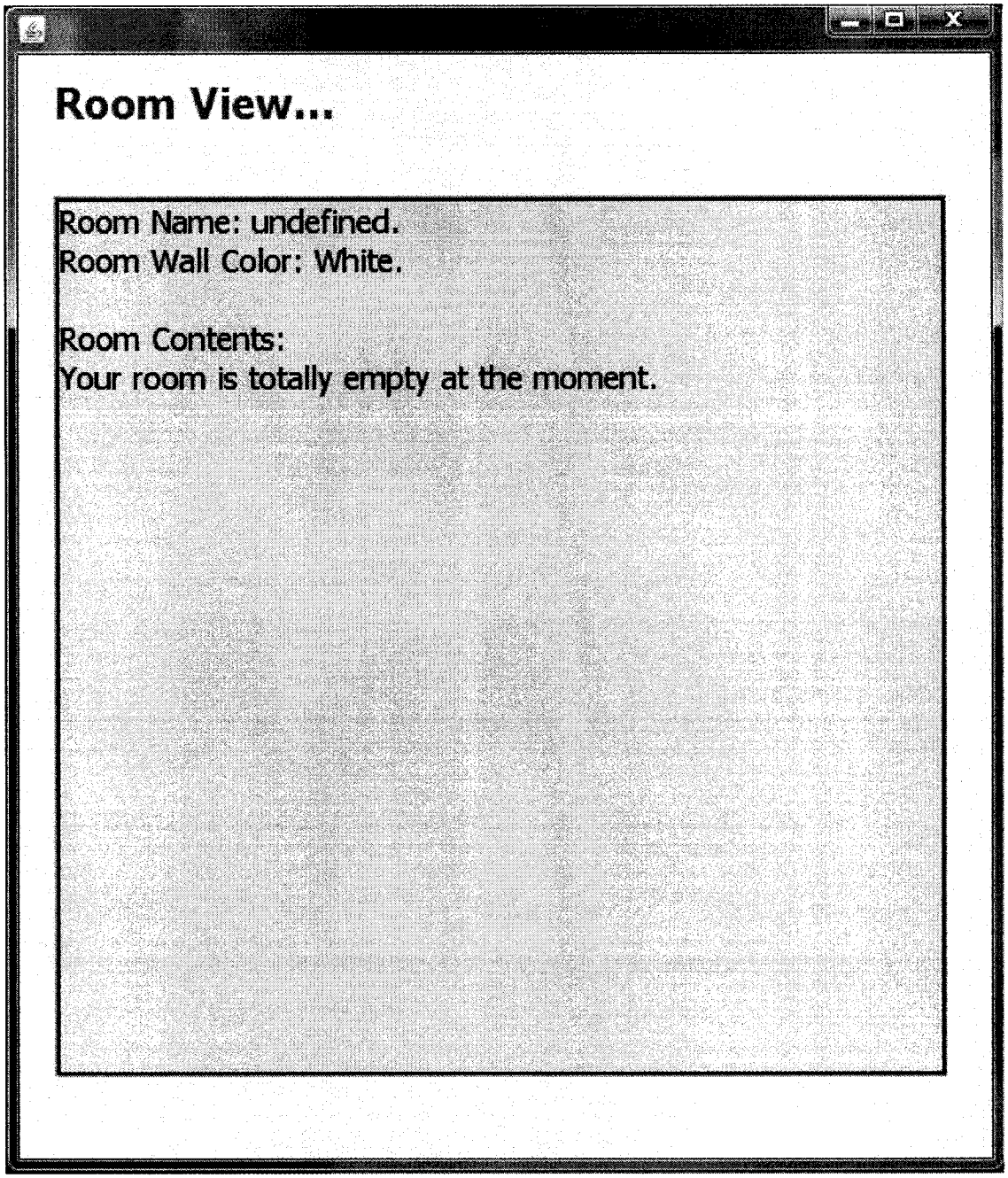

Figure D.2. An empty room view.

After displaying the empty room, users could set the criteria for room personalization such as the maximum number of items allowed in the room or the various

colors that users could choose from to set their room wall color. Figure D.3 shows this thread of actions. Sample code for this window is available in Appendix C.

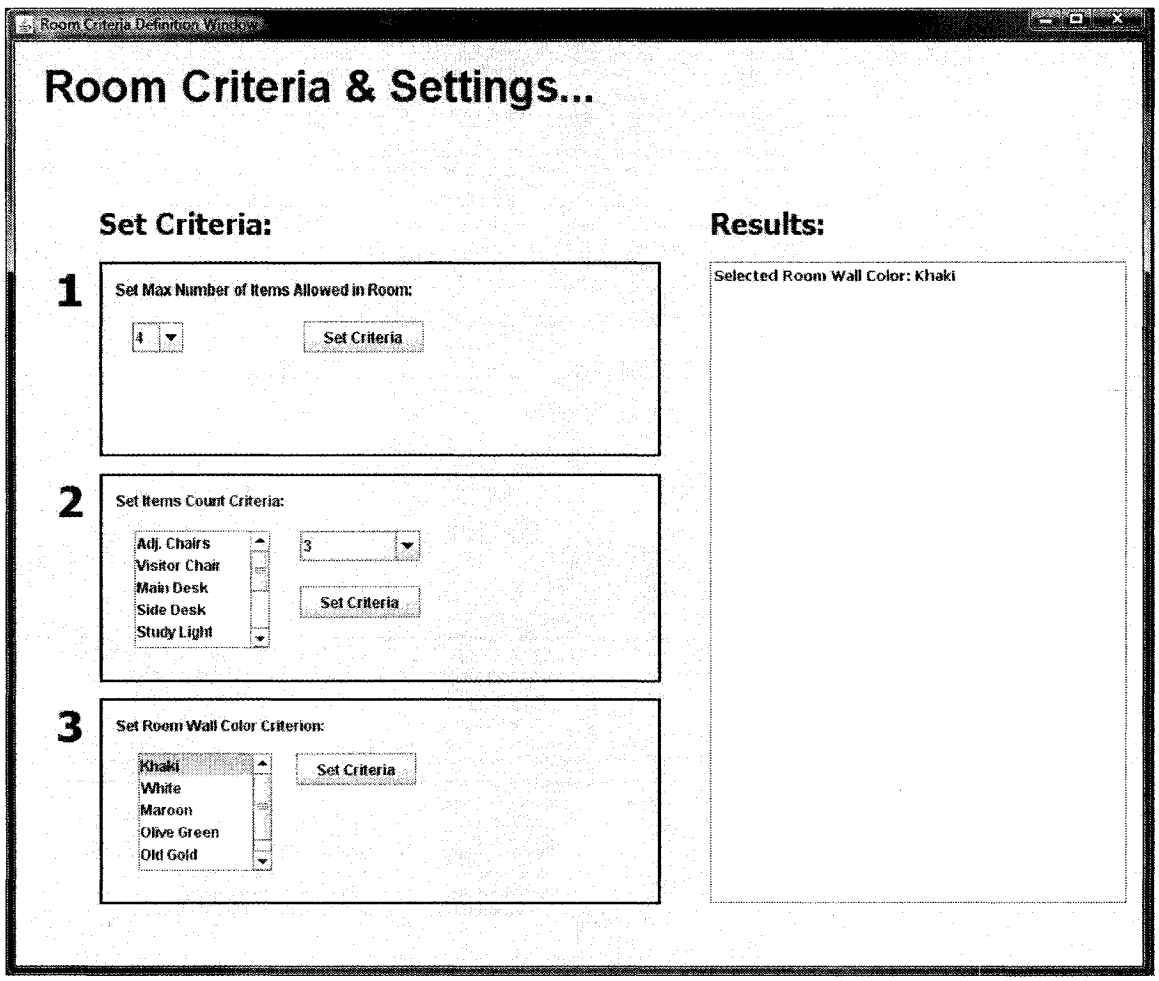

Figure D.3. The room criteria window.

In the Results pane on the right side of the window, all messages associated with the user issued commands are printed. Next, after setting the criteria, the actual personalization of room occurs and the user could launch the window to personalize the room. Figure D.4 depicts the personalization window.

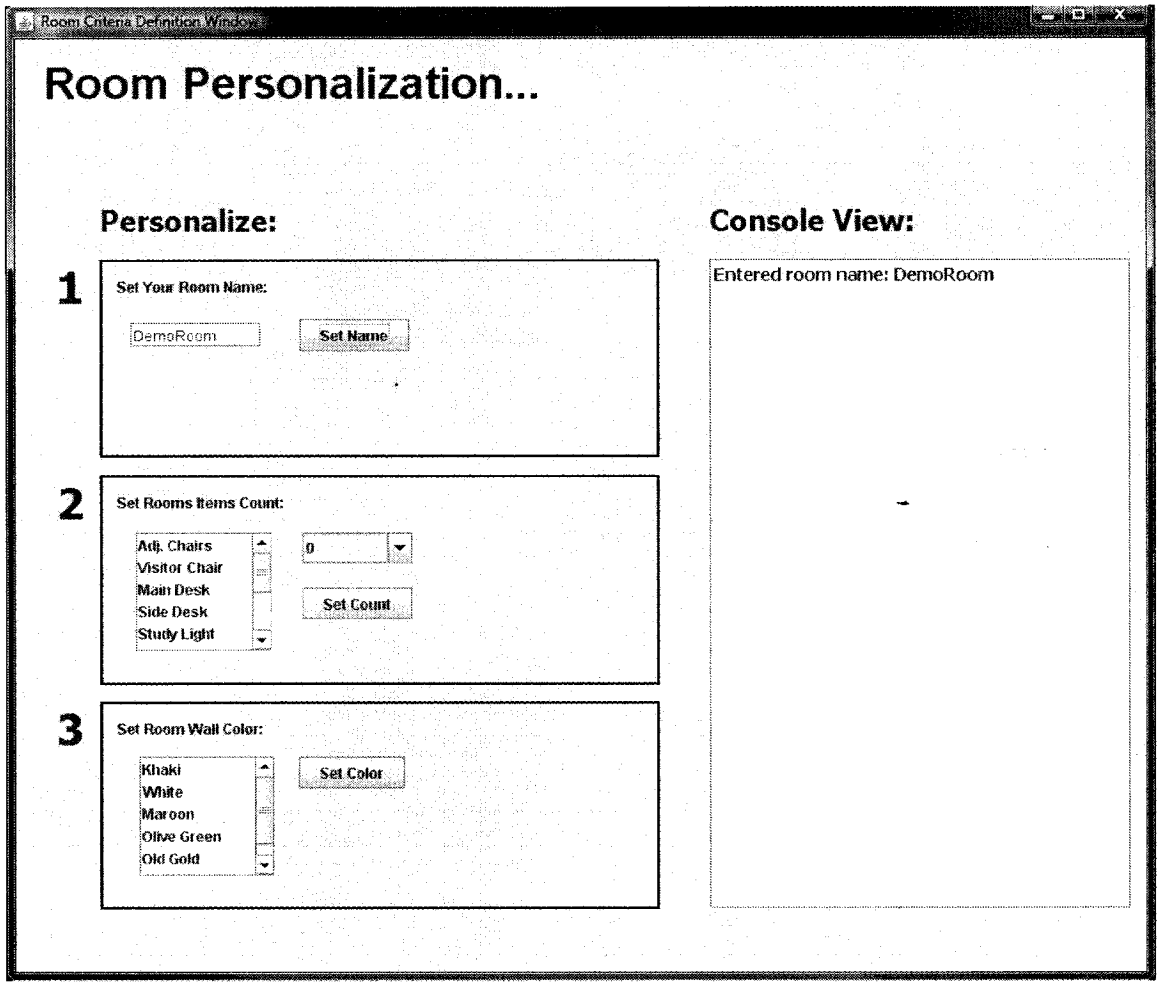

Figure D.4. The room personalization window for a user to personalize a room.

In this demo, the room name is changed to DemoRoom and the room's wall color is changed to khaki from the default white color. The new room view is displayable from the main dashboard. Figure D.5 shows the new changes.

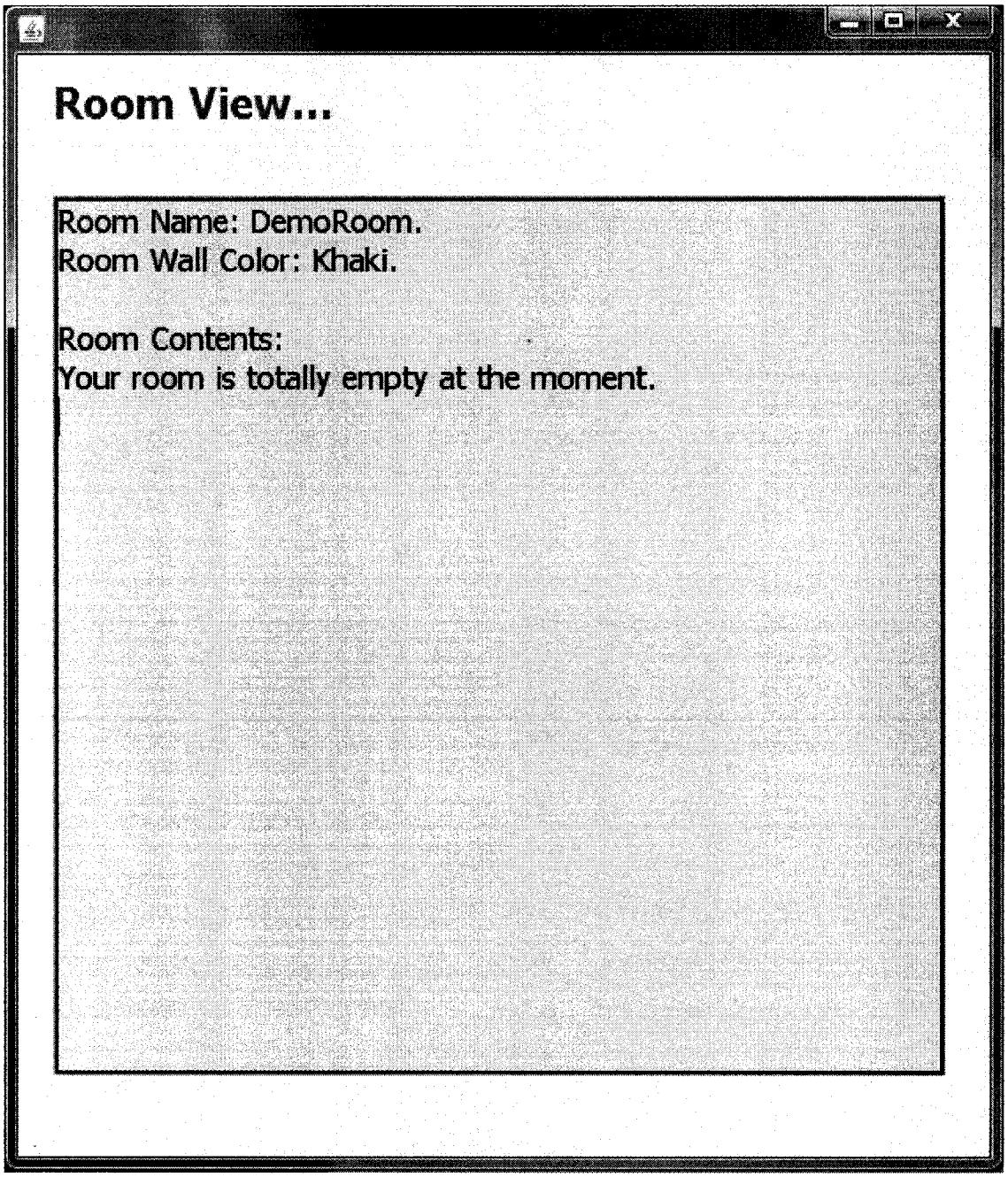

Figure D.5. The room with a name and wall color.

Finally, items could be added to the room using the personalization window. For this demo, four study lights are added to the room. The result is shown in Figure D.6.

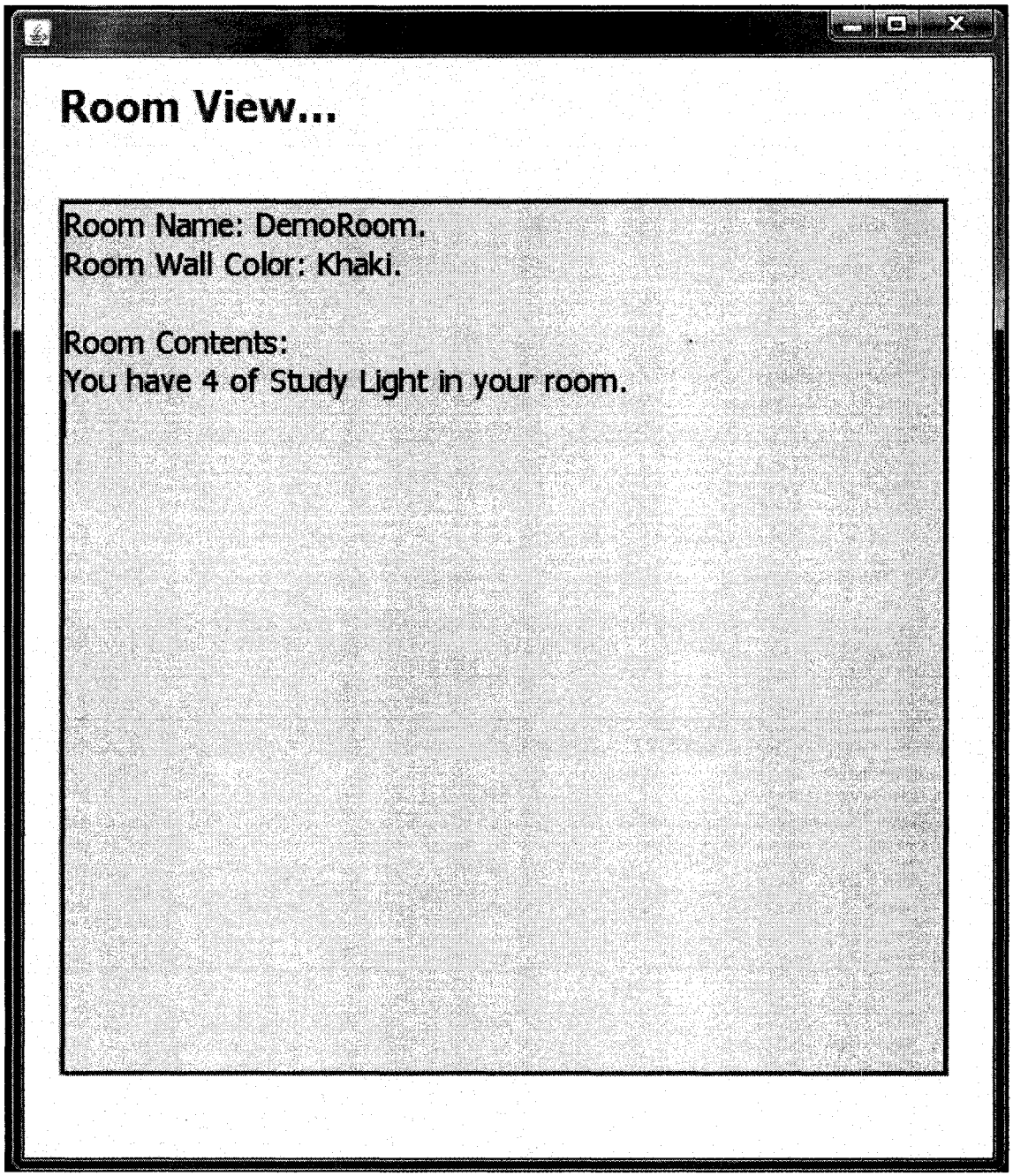

Figure D.6. The new view of room after adding four new study lights.